View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by UPCommons. Portal del coneixement obert de la UPC

TRABAJO DE FIN DE MÁSTER

Máster en Ingeniería Matemática

Título: Firmas de umbral para procesos electorales

Autor: Carlos Luna Mota

Director: María Paz Morillo Bosch

Departamento: MA4

Convocatoria: Junio 2010

Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Índice general

Capítulo 0

Introducción

0.1. Resumen

Los procesos de voto electrónico hacen uso de un gran número de protocolos criptográficos de gran complejidad. La firma digital y, en particular, la firma digital de umbral es una de las componentes básicas de dichos processos.

Pese a que existe una extensa literatura en lo que se refiere a firma digital a nivel teórico, son pocos los estudios que se ocupan de desarrollar los aspectos prácticos de su uso en los procesos electorales y, a fin de suplir esa carencia, se ha llevado a cabo un estudio de las posibilidades que ofrece la firma RSA de umbral en los procesos electorales. Esta memoria pretende resumir y estructurar las conclusiones que se derivan de dicho estudio.

Los objetivos perseguidos en el Trabajo de Final de Master que ha dado lugar a esta memoria son:

- Estudiar el estado del arte en lo referente a firmas RSA de umbral.
- Desarrollar un protocolo de firma RSA que sea distribuido y robusto.
- Determinar la viabilidad de dicho protocolo y sus prestaciones.
- Proponer mejoras de cara a futuras investigaciones en este campo.

La motivación esencial para el uso de firmas RSA en detrimento de otras opciones basadas en, por ejemplo, el logaritmo discreto, es la mayor disponibilidad de hardware especializado en la realización de las operaciones propias de dicho esquema de firma. Sin embargo, es importante tener en cuenta que es necesario valorar el resto de opciones, en especial las que hacen uso del logaritmo discreto y del logaritmo elíptico, antes de llevar a cabo la implementación definitiva del esquema de firma.

0.2. Estado del arte

En general, puede afirmarse que tanto la generación de parámetros RSA como la firma de umbral RSA son problemas resueltos a nivel teórico, si bien los protocolos propuestos en uno y otro caso presentan serias deficiencias a nivel práctico que se han procurado solventar en este proyecto.

A continuación se detalla la base teórica en la que se basan los protocolos propuestos en los capítulos siguientes. Dado que hasta el momento todos los artículos técnicos tratan ambos problemas por separado, es mejor discutir las propuestas también por separado. Sin embargo, es necesario tener en cuenta que las peculiaridades de un protocolo de firma pueden afectar gravemente al protocolo de generación de parámetros RSA y viceversa.

Generación distribuida de parámetros RSA

De los dos problemas a tratar, la generación distribuida de parámetros RSA es, sin duda, el más complicado. Sin ir más lejos, las únicas propuestas realizadas hasta el momento para la generación del módulo RSA N se basan en multiplicar dos números aleatorios de gran tamaño y comprobar a posteriori que ambos son, en efecto, primos. No es necesario decir que se trata de un método altamente ineficiente y complicado, si quiere llevarse a cabo de manera segura y robusta.

La generación de la clave privada d a partir de los valores de N y e es un problema igual de delicado pero mucho menos costoso computacionalmente.

En 1997 Boneh y Franklin [BF97] presentaron un protocolo para generar módulos RSA de manera distribuida que era seguro contra adversarios pasivos, es decir, aquellos que siguen las especificaciones del protocolo pretendiendo obtener información de los participantes honestos. Sin embargo, dicho protocolo no protege contra adversarios activos, que son aquellos que no siempre siguen las especificaciones del protocolo. Así pues, y pese a ser el artículo fundamental que permitió posteriores avances en esta materia, la propuesta de Boneh y Franklin no es suficiente segura como para ser usada en procesos electorales.

En 1998, Frankel, MacKenzie y Yung [FMY98] consiguieron mejorar el protocolo de Boneh y Franklin para hacerlo seguro frente a adversarios activos. Las mejoras incorporadas por estos autores incluían técnicas de compartición de secretos verificable[1], la transformación de la representación de secretos compartidos[2] y herramientas de verificación cruzada[3].

[1] *Verifiable Secret Sharing*
[2] *Share Representation Transformation*
[3] *Cross-checking information*

Todas estas mejoras hacen que el protocolo sea robusto y resistente a una minoría de adversarios activos, lo que significa que siempre finalizará con éxito aún en el caso de que varios usuarios (menos de la mitad) actuen maliciosamente. Ello se consigue detectando a los usuarios maliciosos y eliminándolos del protocolo para, acto seguido, reiniciar el sistema.

Lamentablemente estas mejoras tienen un alto coste computacional, que hay que tener en cuenta a la hora de desarrollar una implementación efectiva del mismo. Concretamente, Frankel et al. aseguran en su artículo que el algoritmo, en el mejor de los casos, tiene un coste 100 veces superior a la versión no robusta de Boneh y Franklin, haciéndolo impracticable en el contexto de un proceso de votación electrónica.

En años posteriores se han presentado nuevas opciones para generar parámetros RSA de manera distribuida pero ninguna de ellas es plenamente satisfactoria. Algesheimer, Camenish y Shoup [ACS02] propusieron un protocolo que produce módulos RSA que son producto de dos primos seguros[4], requisito necesario para algunos protocolos de firma pero no para el que finalmente se utiliza en este proyecto. Esta propuesta es además, algo más ineficiente que la de Frankel et al. de manera que no vale la pena tenerla en cuenta en nuestro caso.

Finalmente Damgård y Mikkelsen [DM10] presentaron recientemente un protocolo de generación de parámetros cuya prueba de primalidad finaliza en una ronda, lo cual es un avance respecto a la propuesta de Frankel et al., que requería unas 80. Lametablemente, en las pruebas preliminares que se realizaron en el transcurso de este proyecto se comprobó que a efectos prácticos ambos tests eran comparables dado que el test de primalidad de Miller-Rabin que usan Damgård y Mikkerlsen es mucho más costoso que el test de Fermat usado por el resto de propuestas.

Firma RSA distribuida

La firma RSA distribuida está, sin duda, mucho más desarrollada que la generación de parámetros. Esto se debe, por un lado, a que resulta mucho más sencillo desarrollar un protocolo seguro y robusto para realizar firmas, y por otro, a que es habitual que una tercera parte de confianza genere los parámetros RSA y distribuya los fragmentos de la clave privada entre los participantes, haciendo innecesario el costoso protocolo de generación distribuida y dotando de interés independiente al protocolo de firma.

La no interactividad es una propiedad siempre deseable en los protocolos distribuidos para evitar cuellos de botella que congestionen la ejecución de los mismos. Así mismo, la no interactividad hace inútiles ataques que pretendan sabotear el proceso mediante la ralentización del protocolo (falta de respuesta, retrasos injustificados, etc...). Es, por lo tanto, muy deseable que el protocolo escogido sea no interactivo además de robusto.

[4] p es un primo seguro si es de la forma $p = 2q + 1$ con q primo.

Shoup [Sho00] presentó en el año 2000 un esquema de firma de umbral RSA que es eficiente, robusto y no interactivo. Hasta ese momento tan sólo se conocían protocolos que, o bien no eran robustos o bien eran interactivos o bien generaban firmas de tamaño excesivo y, por lo tanto, la propuesta de Shoup resulta muy atractiva de cara a su uso práctico.

Sin embargo, dicho protocolo requiere el uso de un módulo RSA formado por dos primos seguros y, si bien existen protocolos que nos permiten obtener ese tipo de parámetros de forma distribuida (como por ejemplo el ya mencionado de Algesheimer, Camenish y el propio Shoup [ACS02]), son varios los motivos que hacen poco recomendable su uso. Sin ir más lejos, resulta mucho más costoso dar con un primo seguro que con un primo normal, cosa que tiene una gran repercusión en el número de iteraciones que serán necesarias para generar los parámetros RSA.

Algo más tarde, Damgård y Koprowski [DK01] presentaron un protocolo de firma RSA de umbral que podía usar un módulo RSA cualquiera. Su propuesta es robusta y no interactiva como la de Shoup pero basa su seguridad en dos hipótesis ad-hoc que, si bien parecen razonables, es mejor evitar. Además, el módulo RSA que usa este protocolo tiene que cumplir ciertos requisitos que supondrían la modificación de los protocolos de generación de parámetros comentados en el apartado anterior.

Afortunadamente en un artículo de Damgård y Dupond [DD05] se propone una nueva versión del protocolo anterior que basa su seguridad en la hipótesis RSA estándar y no impone ninguna condición al módulo RSA usado. Lamentablemente estas mejoras se hacen a costa de perder la no interactividad del protocolo y por lo tanto tampoco se trata de una propuesta plenamente satisfactoria.

A pesar de todo, el protocolo de Damgård y Dupond es la mejor opción hasta el momento y por lo tanto es la que se ha usado como base para este proyecto. En particular, se han añadido las modificaciones necesarias para hacerlo no interactivo a costa de perder eficiencia cuando se trabaja con muchos usuarios (caso que no se contempla en este proyecto).

0.3. Estructura de la memoria

Esta memoria está estructurada de la siguiente manera:

En el capítulo 0 se resume el contenido de este proyecto y se presentan los resultados previos que han dado lugar al mismo. Se explican, además, las convenciones que se seguirán a lo largo de toda la memoria.

En el capítulo 1 se contextualiza el proyecto en el ámbito de los procesos de votación electrónica mientras que en el capítulo 2 se describen las primitivas criptográficas que conforman la base de los protocolos que se describen en el capítulo 3.

Finalmente se dedica el capítulo 4 a comentar los resultados obtenidos y a sugerir posibles mejoras para los protocolos descritos.

Además, se añade un apéndice en el que se comentan las bondades del esquema de compartición de secretos Replicated Integer Secret Sharing a la hora de detectar participantes que actúen de manera fraudulenta durante el proceso de firma.

Representación gráfica de los protocolos

Para expresar con mayor claridad los diferentes protocolos se usarán diagramas MSC[5] que seguirán las siguientes convenciones:

Dada la simetría de los protocolos utilizados, todos ellos se representarán gráficamente desde el punto de vista del participante $i-$ésimo. El resto de participantes quedan representados por un participante genérico j, lo que significa que damos por supuesto que cuando i envía un mensaje a j en realidad está enviando un mensaje **a todo** j mediante comunicaciones seguras punto a punto. Por el contrario, cuando i quiera enviar un mismo mensaje a todos los demás participantes lo hará a través del *canal de broadcast* que inmediatamente lo reenviará a todos los participantes (en la figura 1 se representa este envío mediante lineas discontínuas pero, en general, estas se omitirán). Hay más detalles del canal de comunicación en la sección 2.1.

Los cálculos que efectua cada participante se explicitarán en sus respectivas líneas de ejecución. Las llamadas a subprotocolos se pueden representar de la manera usual (mediante una caja de texto) o de manera transparente (explicitando el subprotocolo). Así mismo se establecerán puntos de control en base a una condición dada. Si no se dice lo contrario se sobreentiende que en caso de cumplirse la condición el protocolo prosigue y en caso de no cumplirse el protocolo se reinicia (con la posible expulsión de algún miembro).

Adicionalmente, se añadirá una flecha discontinua, justo después de la condición, para señalar el punto exacto en el que debe reiniciarse ($\circlearrowleft$) el protocolo.

[5] Message Sequence Charts [MB01]

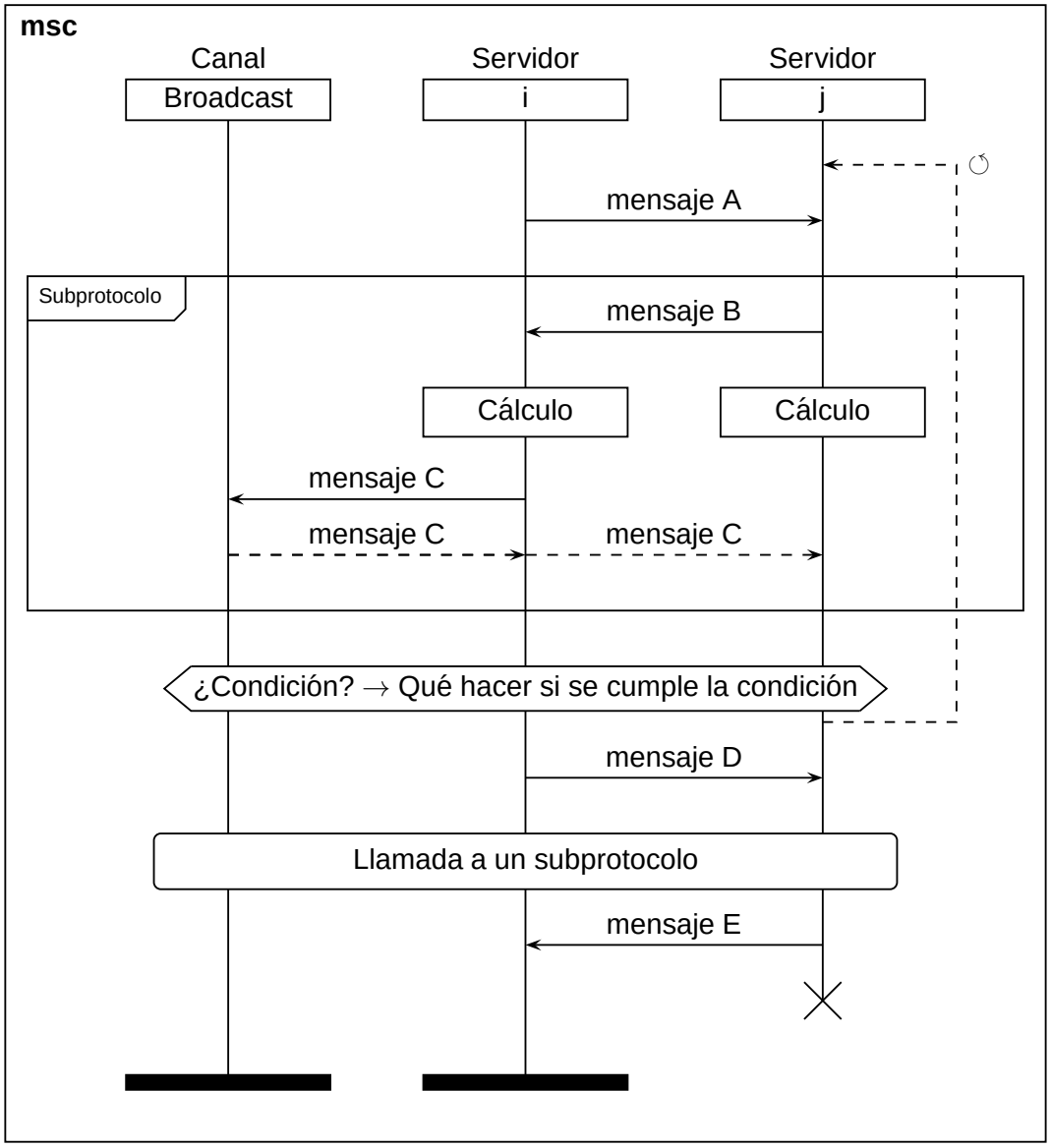


Figura 1: Ejemplo de Diagrama MSC

Finalmente, es posible que algún usuario malicioso sea expulsado del protocolo. Esto se representará mediante unas aspas que, a efectos prácticos, significan el cierre de todo canal de comunicación (tanto punto a punto como broadcast) con dicho usuario por parte de todos los demás usuarios.

0.4. Agradecimientos

Quisiera cerrar esta introducción con unas palabras de agradecimiento para todas aquellas personas que han hecho posible este proyecto.

Empezando por Miquel Soriano, quien confió en mí sin apenas conocerme y por Scytl, la empresa que han hecho viable económicamente dedicar estos meses de mi vida a la investigación.

Y siguiendo con mis compañeros de departamento y de despacho, quienes hicieron agradable mi estancia y me orientaron en todo momento. Mención especial requiere mi compañero Alex Escala, sin el cual este proyecto no hubiese sido posible y al que le debo muchas horas de trabajo y muchos buenos momentos.

También es necesario agradecerle a mi directora de proyecto, Paz Morillo, la confianza que ha depositado en mí, los sabios consejos que me ha regalado y el ambiente inolvidable que he podido disfrutar durante todos estos meses.

Por último es preciso agradecerle a mi familia y a mi pareja la paciencia que han tenido conmigo y el esfuerzo que han realizado, a lo largo de los años, para ayudarme a llegar hasta aquí.

A todos los mencionados y a muchos otros amigos cuyos nombres no están presentes en esta página pero sí en mi memoria, ¡Muchas gracias!

Capítulo 1

Votación electrónica

Tradicionalmente se asocia la criptografía al cifrado y descifrado de mensajes sensibles que quieren mantenerse en secreto. Sin embargo, en la era de las telecomunicaciones, la criptografía se ha convertido en una herramienta ubícua que se encarga de multitud de aspectos.

Los procesos de voto electrónico son, sin duda, una de las aplicaciones más compleja y sensible de la criptografía. Demostrar que las elecciones se llevan a cabo de manera justa y equitativa mientras se garantiza la privacidad del votante es una tarea que requiere herramientas criptográficas de gran complejidad y grandes precauciones a la hora de usarlas. No en vano se trata de un campo en plena expansión y que aún genera mucha desconfianza.

A pesar de todo ello, resulta totalmente evidente que el futuro de los procesos electorales pasa por digitalizar dichos procesos, solventando así los problemas logísticos asociados y haciéndolos mucho más accesibles al conjunto del electorado. La digitalización de los procesos electorales permitirá, así mismo, alcanzar mayores cotas de participación ciudadana en la toma de decisiones con la consiguiente mejora de la calidad democrática de los países que adopten dicha medida.

1.1. Requisitos de seguridad

Como ya se ha dicho, los procesos de votación electrónica son especialmente sensibles desde el punto de vista criptográfico y, por lo tanto, es necesario que todo protocolo relacionado con ellos cumpla los estrictos requisitos de seguridad que se detallan a continuación:

Autenticidad del votante: Debe garantizarse que tan sólo votarán aquellas personas que tengan derecho a voto y que éstas no emitirán más de un voto.

Privacidad del votante: Debe garantizarse que es imposible relacionar a un votante con su voto, manteniendo así en secreto la elección que ha hecho el votante.

Precisión en los resultados: Los resultados obtenidos deben reflejar fielmente los votos emitidos de manera legítima, impidiendo que se añadan, se borren o se modifiquen votos de manera ilegítima.

Privacidad de los resultados intermedios: Debe ser imposible obtener información alguna de las votaciones hasta que no termine todo el proceso para evitar influenciar a los votantes que aún no han ejercido su derecho.

Verificabilidad del voto: Cada votante debe poder verificar de forma independiente que su voto ha sido incluido en el recuento final.

No coacción: Debe ser imposible para un votante demostrar el contenido de su voto para evitar casos de coacción o compra de votos.

No es en absoluto obvio cómo pueden compatibilizarse todos estos requisitos y en muchos casos es necesario hacer uso de herramientas criptográficas sofisticadas a tal efecto. Es habitual, por ejemplo, hacer uso de computaciones distribuidas para repartir las responsabilidades y evitar que un único usuario corrupto comprometa la seguridad del protocolo.

Así, un proceso electoral electrónico podría seguir la siguiente estructura: Los votos se emiten cifrados con la clave pública de la mesa electoral y firmados con el certificado digital de cada votante (asegurando su privacidad y su autenticidad). Una vez que los votos llegan a la mesa electoral, que está formada por unas 10 personas aproximadamente, ésta se encarga de verificar que las firmas corresponden a personas con derecho a voto para, acto seguido, descifrar cada voto, recontarlos en paralelo y firmar la lista de votos descifrados[1]. Este proceso se lleva a cabo de manera distribuida para que la responsabilidad no recaiga en una única persona y de él depende en gran parte la seguridad de todo el protocolo de voto electrónico.

En esta memoria se estudia un protocolo para generar firmas digitales de umbral, que es la estructura de acceso más habitual en este tipo de procesos. En todo momento se han tenido en cuenta los parámetros habituales de una mesa electoral (10 o menos usuarios, comunicaciones en red local seguras y fiables, hardware disponible, ...) y las necesidades propias de un proceso electoral (duración breve, máxima seguridad, sin hacer uso de terceras partes de confianza, ...). Por todo ello se ha optado por una firma RSA, para la que se disponde de hardware especializado de alta velocidad y se han adaptado los algoritmos teóricos existentes hasta el momento, cambiando algunos procedimientos por otros que son igual de seguros pero mucho más eficientes en el rango en el que nos movemos.

[1] Los protocolos basados en esquemas de cifrado homomórficos no requieren descifrar cada voto para hacer el recuento. Aún así es frecuente que la mesa electoral firme, de manera conjunta, el resultado.

Capítulo 2

Primitivas criptográficas

2.1. Canal de comunicación

A la hora de implementar un protocolo criptográfico es necesario hacer alguna suposición respecto al hardware del que dispondrán tanto los usuarios como los adversarios. En particular, el canal de comunicación es un punto de especial relevancia en los protocolos interactivos y por ello es conveniente aclarar aquí algunas suposiciones que se hacen al respecto.

Punto-a-punto

En un protocolo de computación distribuida es frecuente usar comunicaciones seguras punto-a-punto entre usuarios. Es decir, asumimos que existe algun tipo de infraestructura real o virtual que nos asegura el envío de mensajes entre usuarios con las siguientes propiedades:

Confidencialidad: Tan sólo el receptor del mensaje será capaz de leer su contenido.

Autenticidad: Nadie puede enviar mensajes en nombre de otra persona.

Integridad: Una vez enviado un mensaje no podrá ser modificado.

No-Repudio: El autor de un mensaje no puede negar su autoría.

Estas propiedades pueden implementarse de manera física (con hardware auditado de alta seguridad y conexiones punto a punto) o de manera virtual (con, por ejemplo, una infraestructura de clave pública que permita cifrar y firmar mensajes). Si bien la primera opción es, en general, mucho más eficiente, en algunos contextos resulta inviable por motivos económicos (cuando una infraestructura física no pueda ser reutilizada) o prácticos (cuando los participantes trabajen de manera remota).

Demoras injustificadas

Además de las propiedades anteriormente citadas es necesario exigir al canal un tiempo de respuesta acotado y consistente. Es decir, el mensaje debe llegar siempre a su destino y debe hacerlo en un intervalo de tiempo razonable. Todo ello se exige para evitar que un participante malicioso pueda afirmar que ha enviado un mensaje que en realidad no ha enviado o, incluso, detenga el avance del protocolo.

Este último punto está muy relacionado con la fiabilidad del hardware usado en el canal de comunicación y puede gestionarse mediante subrutinas que señalen como maliciosos a los usuarios que demoren el envío de sus mensajes más allá de cierto límite *razonable*.

No es sencillo establecer un límite a priori para el envío de mensajes porque éste depende tanto de la capacidad de cálculo de los diferentes participantes como de las propiedades del canal. Resulta imprescindible pues determinar en cada caso cuál debe ser el criterio de detección de usuarios maliciosos.

Directorio público y canal de broadcast

Por último es necesario fijar nuestra atención en que algunos pasos del protocolo distribuido de Generación de Párametros RSA (ver sección 3.1) requieren un mecanismo de *broadcast* que permita a un usuario enviar, de manera eficiente, un mensaje a todos los demás usuarios con los mismos requisitos exigidos a la comunicación punto-a-punto.

Existen diversos protocolos que resuelven el problema de los Generales Bizantinos [LSP82], uno de cuyos casos particulares es el broadcast, como por ejemplo el propuesto en [BTHR07], pero estos suelen ser bastante ineficientes en general y a efectos prácticos es más útil utilizar un esquema de *directorio público*[1] que permite a los usuarios publicar mensajes y leer lo que otros han publicado pero no borrar o modificar ningún mensaje (ni siquiera los propios). El directorio público debe asegurar, así mismo, la identidad de los autores de los mensajes.

Por último, el directorio público puede servir para obtener información del exterior. A la hora de usar un esquema de compromiso (ver sección 2.3) es necesario disponer de algunos parámetros que deberán ser calculados por una tercera parte de confianza (lo que se conoce como *Common Reference String Model*). Nótese que estos parámetros no permiten a dicha tercera parte obtener información del protocolo y, por lo tanto, no ponen en peligro la seguridad del mismo ni entran en conflicto con su naturaleza distribuida.

[1] En inglés suele usarse la denominación *Public Board*

2.2. Compartición de secretos

La compartición de secretos es una primitiva fundamental en el ámbito de la computación distribuida. En nuestro protocolo usaremos diversos esquemas de compartición de secretos en función de la naturaleza de la información a compartir y la estructura de acceso requerida en cada caso.

2.2.1. Estructuras de acceso

Una estructura de acceso, Γ, asociada a un conjunto $C = \{x_1, \ldots, x_n\}$ es un conjunto de subconjuntos de C que denominaremos *subconjuntos autorizados*, $\Gamma \subset \mathcal{P}(C)$.

En general, las estructuras de acceso se usan para determinar qué subconjuntos de usuarios están autorizados a realizar una determinada acción. En el caso particular de la compartición de secretos la estructura de acceso determina qué subconjuntos de usuarios están autorizados a recuperar el secreto si colaboran mutuamente.

Es importante notar que en este ámbito hablaremos siempre de *estructuras de acceso monótonas*, que son aquellas tales que si $A \in \Gamma$ y $A \subset B$ entonces $B \in \Gamma$.

Estructuras de acceso de umbral

Las *estructuras de acceso de umbral* son una familia especialmente importante de estructuras de acceso monótonas. Se definen en función de un parámetro t y son de la forma: $\Gamma_t = \{A \subset C \mid \#A \geqslant t\}$.

Es decir, todos los subconjuntos de C que contengan al menos t miembros son subconjuntos autorizados. Esto se suele denominar habitualmente esquema de acceso t-de-n, siendo $n = \#C$.

Existen dos casos particulares de estructura de acceso de umbral que es necesario destacar: el esquema $\lceil \frac{n+1}{2} \rceil$-$de$-$n$, o *mayoría simple*, y el esquema n-de-n, o *unanimidad*.

Obsérvese que en el esquema de mayoría simple, se requiere que los subgrupos autorizados sean estrictamente mayores que $\frac{n}{2}$. Es, por lo tanto, un esquema óptimo para protocolos distribuidos que pretendan ser seguros y robustos, permitiendo el máximo número de usuarios maliciosos ($n - t$) sin que la seguridad del mismo se vea comprometida ($t > n - t$).

2.2.2. Compartición de secretos de Shamir

En 1979, Adi Shamir [Sha79] publicó un método para compartir secretos con una estructura de acceso del tipo t-de-n que hoy es estándar. Existen, de hecho, numerosas

adaptaciones de su esquema original y a continuación se detallan las más relevantes para este proyecto.

El esquema original permite compartir un elemento s de $\mathbb{Z}_q$ (con q primo) mediante un polinomio de grado $t-1$ de la forma:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$$

con el término independiente igual al secreto a compartir, $a_0 = s$, y el resto de coeficientes elegidos de manera independiente, aleatoria y uniforme, $a_i \in_R \mathbb{Z}_q$.

Para compartir el secreto s es necesario que un *repartidor de confianza*[2] genere el polinomio $p(x)$ y envíe a cada usuario el par (i, s_i) donde i es el identificador único de dicho usuario (típicamente: $1, 2, 3, \ldots, n$) y $s_i = p(i)$.

Para recuperar el secreto, un subgrupo S de al menos t miembros puede interpolar el polinomio $p(x)$ a partir de los t pares (i, s_i) usando interpolación de Lagrange:

$$s = a_0 = p(0) = \sum_{i \in S} s_i \lambda_i^S(0)$$

donde $\lambda_i^S(x) = \prod_{\substack{j \in S \\ j \neq i}} \frac{j-x}{j-i}$

Reconstruyendo secretos en el exponente

Podemos usar una variante de dicho esquema para reconstruir un secreto *en el exponente*:

$$m^s = \prod_{i \in S} m_i^{\lambda_i^S(0)}$$

donde $m_i = m^{s_i} = m^{p(i)}$

Para el caso particular n-de-n existe un esquema más eficiente que el propuesto por Shamir. Consiste en repartir el secreto aditivamente entre los participantes:

$$s = \sum_{i=1}^{n} s_i$$

[2] En la literatura criptográfica suele usarse la denominación inglesa: *Trusted Dealer*

Este esquema también permite recuperar secretos en el exponente:

$$m^s = \prod_{i=1}^{n} m_i$$

Otra de sus ventajas es la capacidad para controlar errores de redondeo a la hora de operar con el secreto compartido:

$$\frac{s}{k} = \sum_{i=1}^{n} \lfloor \frac{s_i}{k} \rfloor + \varepsilon$$

donde k es un divisor de s y $\varepsilon \leq n$.

Por todo ello a veces es conveniente usar este esquema cuando se trabaja con una estructura de acceso n-de-n. A fin de poder pasar de una compartición de secretos del tipo Shamir a un esquema aditivo se usarán dos mecanismos de conversión: el *sum-of-poly*[3] y el *poly-to-sum*[4].

Sum-of-poly

Realizando un sum-of-poly es posible transformar un secreto compartido aditivamente en un secreto compartido mediante un esquema de Shamir t-de-n.

$$s = \sum_{i=1}^{n} s_i$$

Cada participante debe generar un polinomio aleatorio $p_i(x)$ de grado $t-1$ tal que $p_i(0) = s_i$ y, a continuación, enviar el valor $s_{i,j} = p_i(j)$ al usuario $j-$ésimo mediante una conexión segura punto a punto (sin olvidarse de guardar para sí el valor $s_{i,i}$).

Una vez que todo el mundo ha enviado y recibido dichos valores cada usuario debe computar:

$$s'_i = \sum_{j=1}^{n} s_{j,i}$$

donde las s'_i cumplen la relación $s'_i = p(i)$ siendo $p(x)$ un polinomio aleatorio de grado $t-1$ tal que $p(0) = s$. El polinomio $p(x)$ puede representarse como $p(x) = \sum_{i=1}^{n} p_i(x)$.

Es fundamental observar que en este caso no hemos necesitado un repartidor de confianza para repartir un secreto entre los participantes y que éste es, en general, un

[3] Del Inglés: *suma de polinomios*
[4] Del Inglés: *de polinomio a suma*

buen método para generar de manera distribuida un valor s y repartirlo entre un grupo de usuarios siguiendo un esquema t-de-n tal y como se explica en [GJKR01].

Poly-to-sum

Realizar un poly-to-sum [FGMY97] es mucho más sencillo que realizar un sum-of-poly dado que para pasar de un esquema Shamir t-de-n a un esquema aditivo basta con multiplicar cada s_i por el coeficiente de Lagrange adecuado.

En particular:

$$s'_i = s_i \cdot \lambda_i^C(0) = s_i \cdot \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{j}{j-i}$$

2.2.3. Shamir sobre los enteros

Hasta ahora se han discutido mecanismos de compartición de secretos pertenecientes al cuerpo finito $\mathbb{Z}_q$. Sin embargo, todas estas técnicas pueden usarse en $\mathbb{Z}$ aplicando las pequeñas modificaciones que se explican a continuación.

El principal problema de trabajar en $\mathbb{Z}$ es no poder realizar inversos en el exponente. Para solventarlo es necesario añadir, en algunos puntos, un factor $\Delta(= n!)$ que nos asegure que el resultado de todas las divisiones será siempre entero. Hay al menos dos maneras equivalentes de hacer esto. En [FGMY97] se propone generar los coeficientes del polinomio de la siguiente manera:

$$m^s = \prod_{i \in S} m_i^{\lambda_i^S(0)}$$

$$a_0 = \Delta^2 \cdot s$$
$$a_i \in_R \{0, \Delta, 2\Delta, 3\Delta, \ldots, \Delta^3 K^2\}$$

donde s se elige en el intervalo $[0, K]$

En [DD05] se usa un método ligeramente diferente para reconstruir el secreto con un esquema t-de-n en el exponente:

$$m_i = m^{2\Delta s_i}$$
$$m^{4\Delta^2 s} = \prod_{i \in S} m_i^{2\Delta \lambda_i^S(0)}$$

donde $s_i = p(i)$ y $\lambda_i^S(x)$ son los coeficientes de Lagrange habituales.

Otro problema relacionado con la elección de los coeficientes de los polinomios (lo que incluye a_0, que es el secreto a compartir) es cómo conseguir que ésta sea aleatoria y uniforme en $\mathbb{Z}$. Para solucionarlo se elige un intervalo finito (pero suficientemente grande) sobre el que escogeremos dichos coeficientes. Dicho intervalo dependerá de un parámetro de seguridad que marcará la distancia estadística entre los coeficientes de un polinomio $p_1(x)$ tal que $p_1(0) = s$ y los coeficientes de otro polinomio $p_2(x)$ tal que $p_2(0) = 0$ como se señala en [DD05].

2.3. Compromisos

La otra primitiva imprescindible para llevar a cabo cualquier computación distribuida en presencia de adversarios activos es el compromiso de valores.

Un *compromiso* es una primitiva que impide a un participante malicioso cambiar los valores con los que está trabajando sin que el resto de participantes lo sepa.

Lanzamientos de monedas virtuales

El ejemplo más común de compromiso es el lanzamiento de moneda virtual o *Coin Flipping*. Los participantes i y j quieren elegir un bit b de manera aleatoria y conjunta. Para ello basta con que cada cual escoja un bit (b_i, b_j) para, acto seguido, combinarlos mediante una operación XOR. Si ambos bits se han elegido de manera uniformemente aleatoria dicha operación garantiza la obtención de un bit que es, así mismo, uniformemente aleatorio.

Sin embargo el participante j podría esperar hasta haber recibido el bit b_i y fabricar entonces un bit b_j tal que el resultado final, b, le favorezca. Para evitar esta estrategia se puede usar una tercera parte de confianza que intercambie los valores de manera honesta una vez recibidos ambos, pero este nuevo protocolo tan sólo traslada el problema de la fiabilidad a un nuevo participante, introduciendo mayor complejidad sin apenas mejorar la seguridad.

Gracias a los esquemas de compromiso es posible realizar un lanzamiento de moneda virtual sin necesidad de una tercera parte de confianza. Para ello se sigue el protocolo descrito en la figura 2.1.

Si ambos participantes siguen fielmente el protocolo al final del mismo habrán intercambiado sus respectivos bits aleatorios (*Completeness*). Mientras que si alguno de los dos intenta mentir el protocolo debería alertar de ello al otro (*Soundness*).

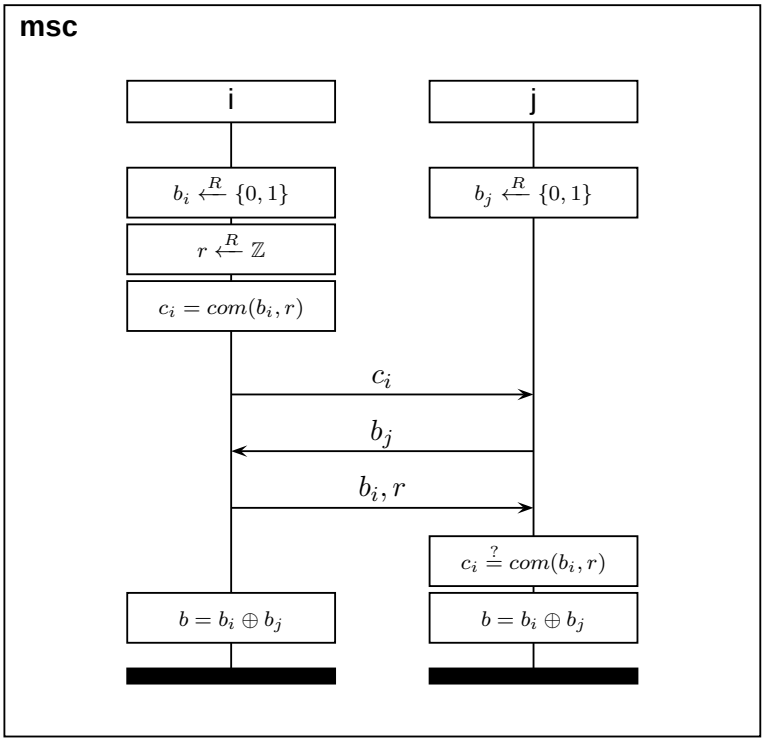


Figura 2.1: Lanzamiento de moneda virtual

Para que se cumpla esta última condición es necesario que el esquema de compromiso utilizado tenga dos propiedades básicas:

Hiding Un esquema es hiding si es imposible distinguir entre $com(x, r)$ y $com(x', r)$ para todo par de posibles valores, x, x' y para todo valor aleatorizador r.

Binding Un esquema es binding si, para un par dado (x, r), es imposible encontrar otro par (x', r') tal que $x \neq x'$ y $com(x, r) = com(x', r')$.

La primera propiedad nos asegura que el participante que recibe el compromiso no puede hacer trampas dado que es imposible para él recuperar el valor x. La segunda nos asegura que el participante que emite el compromiso no puede cambiar el valor comprometido, x, a posteriori. Es imposible encontrar un esquema de compromiso que cumpla las dos propiedades de manera ideal dado que si los conjuntos de compromisos para un valor dado $C_x = \{com(x, r)\}_r$ son disjuntos dos a dos ($\forall\ x \neq x'\ C_x \cap C_{x'} = \emptyset$) entonces bastaría con examinar todos los posibles compromisos para distinguirlos entre sí (perdiendo la propiedad de *hiding*). Si, por el contrario existe alguna intersección, entonces se pierde la propiedad de *binding*.

Por todo ello suele hablarse de compromisos que son perfectamente *hiding* y computacionalmente *binding* o de compromisos que son computacionalmente *hiding* y perfectamente *binding*. A efectos prácticos se trata de una distinción poco relevante dado que

se pueden diseñar esquemas de compromiso tales que la probabilidad de éxito de un usuario malicioso sea arbitrariamente pequeña.

2.3.1. Compromisos de Pedersen

Pedersen introduce en [Ped91] un esquema de compromiso orientado a la Verificación de Secretos Compartidos que es perfectamente *hiding* y computacionalmente *binding* (asumiendo que el cálculo de logaritmos discretos es un problema difícil).

En dicho esquema se trabaja con elementos de $\mathbb{Z}_q$, que es el subgrupo de $\mathbb{Z}_p^*$ de orden q (con p y q primos de gran tamaño tales que q divide a $p-1$).

Concretamente, si g es un generador de $\mathbb{Z}_q$ y $h = g^\nu$ para algún $\nu \in \mathbb{Z}_q$ desconocido tanto por el usuario que genera el compromiso como por el que lo recibe, entonces:

$$com(x,r) = g^x h^r \mod p$$

donde x es el valor de $\mathbb{Z}_q$ que queremos comprometer y r es el elemento de $\mathbb{Z}_q$ que aleatoriza el resultado.

Dado que g, h y p son públicos, para *abrir* el compromiso basta con revelar el par (x, r) y comprobar que, en efecto, $com(x,r) = g^x h^r \mod p$.

Este esquema es interesante por, entre otras propiedades, permitir la verificación de secretos compartidos (ver sección 2.3.3) gracias a sus propiedades homomórficas[5].

2.3.2. Compromisos de Damgård–Fujisaki

Tal y como pasa con los esquemas de compartición de secretos (ver sección 2.2.3), los compromisos de Pedersen también pueden extenderse para compartir elementos de $\mathbb{Z}$. Damgård y Fujisaki proponen en [DF02] el método para realizar compromisos que se describe a continuación:

$$com(x,r) = g^x h^r \mod N_c$$

donde N_c es un módulo RSA (producto de dos primos de gran tamaño), g es un elemento de orden grande de $\mathbb{Z}_{N_c}$ y $h = g^\nu$ para algún $\nu \in_R [0, 2^{3 \log N}]$ desconocido tanto por el usuario que genera el compromiso como por el que lo recibe.

Los compromisos de Damgård–Fujisaki conservan muchas de las propiedades útiles de los compromisos de Pedersen siempre que los elementos que intervienen en ellos ($g, h, x, r, \ldots$) se elijan en el intervalo adecuado (ver [DF02]).

[5] Decimos que un esquema de compromiso es *homomórfico* si cumple la relación: $com(x+x', r+r') = com(x,r) \cdot com(x', r')$

2.3.3. Verificación de Secretos Compartidos

Como se ha visto en la sección 2.2.2, para compartir un valor a_0 podemos usar un polinomio aleatorio de la forma:

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$$

Pues bien, usando un segundo polinomio del mismo grado, $a'(x) = \sum a'_k x^k$, donde a_k se escoge de manera aleatoria uniforme para todo k, es posible generar los compromisos necesarios, $\alpha_0, \ldots, \alpha_{t-1}$, para probar que los fragmentos del secreto repartidos a los demás usuarios han sido correctamente calculados.

Así pues, para compartir el valor a_0 se enviará al jugador $j-$ésimo el par ($a(j)$, $a'(j)$) y se publicarán los compromisos $\alpha_k = g^{a_k} h^{a'_k} \mod p$ para que cada cual pueda comprobar que se cumple la relación:

$$g^{a(j)} h^{a'(j)} = \prod_{k=0}^{t-1} \alpha_k^{(j^k)} \mod p$$

Es preciso observar que los valores α_k no ofrecen información alguna al resto de usuarios pero sí permiten verificar que los elementos $a(j)$ y $a'(j)$ son evaluaciones de polinomios de grado $t-1$.

2.3.4. Pruebas de conocimiento

En la sección 2.3.1 se ha hablado de la posibilidad de *abrir* un compromiso $com(x, r)$ enviando los valores x y r al receptor del mismo. Sin embargo, en muchas ocasiones será necesario demostrar al verificador que poseemos dichos valores sin que éste pueda obtener información alguna sobre ellos.

A toda prueba de conocimiento se le deben exigir tres propiedades fundamentales:

Completeness Si el usuario dispone de la información que dice conocer, entonces la prueba debe ser superada.

Soundness Si el verificador no dispone de la información que dice conocer, entonces la prueba no debe superarse.

Conocimiento cero El verificador no puede obtener información alguna respecto a los valores que el usuario está poniendo a prueba.

La propiedad de *soundness* no se cumple de manera absoluta en algunos pruebas de conocimiento cero. Cuando ése sea el caso hablaremos del *error de soundness*, es decir, la probabilidad de que un usuario engañe al verificador.

Pruebas de conocimiento cero de un valor comprometido

En [DF02] se propone el protocolo descrito en la figura 2.2 para demostrar que, en efecto, el autor de un compromiso $com(x, r)$ conoce los valores x y r sin necesidad de darlos a conocer.

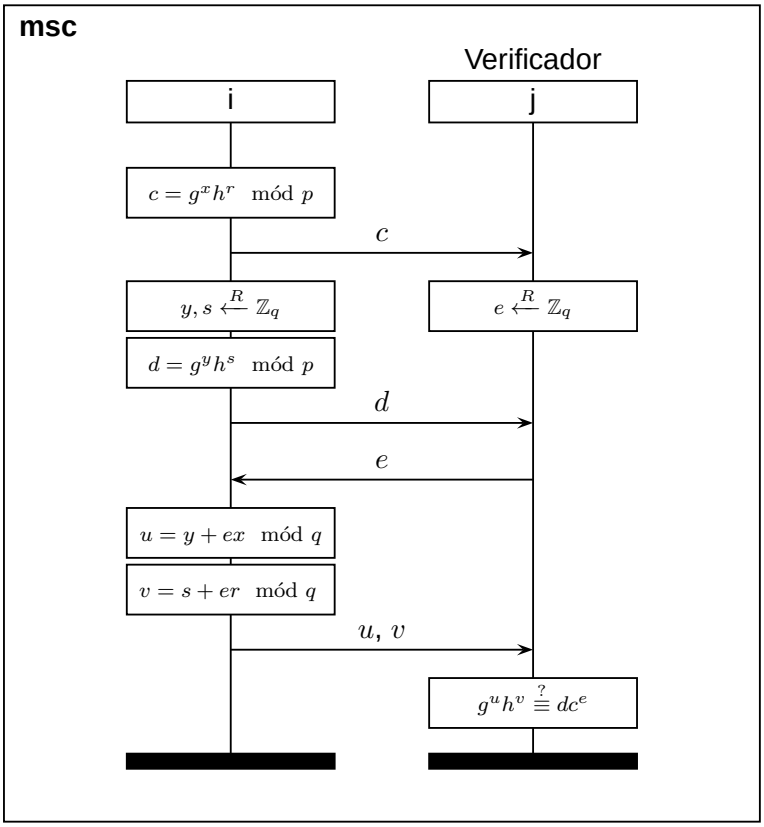


Figura 2.2: Prueba de conocimiento de un valor comprometido

Dicho protocolo consiste en generar un par de valores aleatorios y, s y el correspondiente compromiso[6] $d = g^y h^s$. Acto seguido se usa el valor aleatorio e que ha generado el verificador para calcular $u = y + ex \mod q$ y $v = s + er \mod q$.

Una vez que el verificador recibe los valores u y v tan sólo tiene que comprobar que $g^u h^v = dc^e$.

Pruebas de conocimiento cero de igualdad de logaritmos

Otra prueba de conocimiento cero que se usará en el protocolo de generación y firma RSA distribuida será la que permite a un usuario demostrar que conoce los valores x, y y z tales que $c_1 = g^x h^y$ y $c_2 = \tilde{g}^x \tilde{h}^z$ donde c_1, c_2, g, h, $\tilde{g}$ y $\tilde{h}$ son conocidos pero, no interesa revelar los exponentes.

[6] De Pedersen

La prueba de conocimiento descrita en la figura 2.3 tiene un error de soundness del 50 %, es decir, en cada prueba, el usuario que quiere demostrar que conoce los valores x, y y z tiene un 50 % de probabilidades de engañar al verificador. Sin embargo, repitiendo las pruebas κ veces de manera independiente la probabilidad de engaño se reduce a $2^{-\kappa}$. A efectos prácticos con $\kappa = 40$ habrá suficiente.

Cada una de las κ pruebas consiste en generar una tripleta de números aleatorios q, r, s y enviar al verificador los compromisos[7] $g^q h^r \mod N$ y $\tilde{g}^q \tilde{h}^s \mod N$.

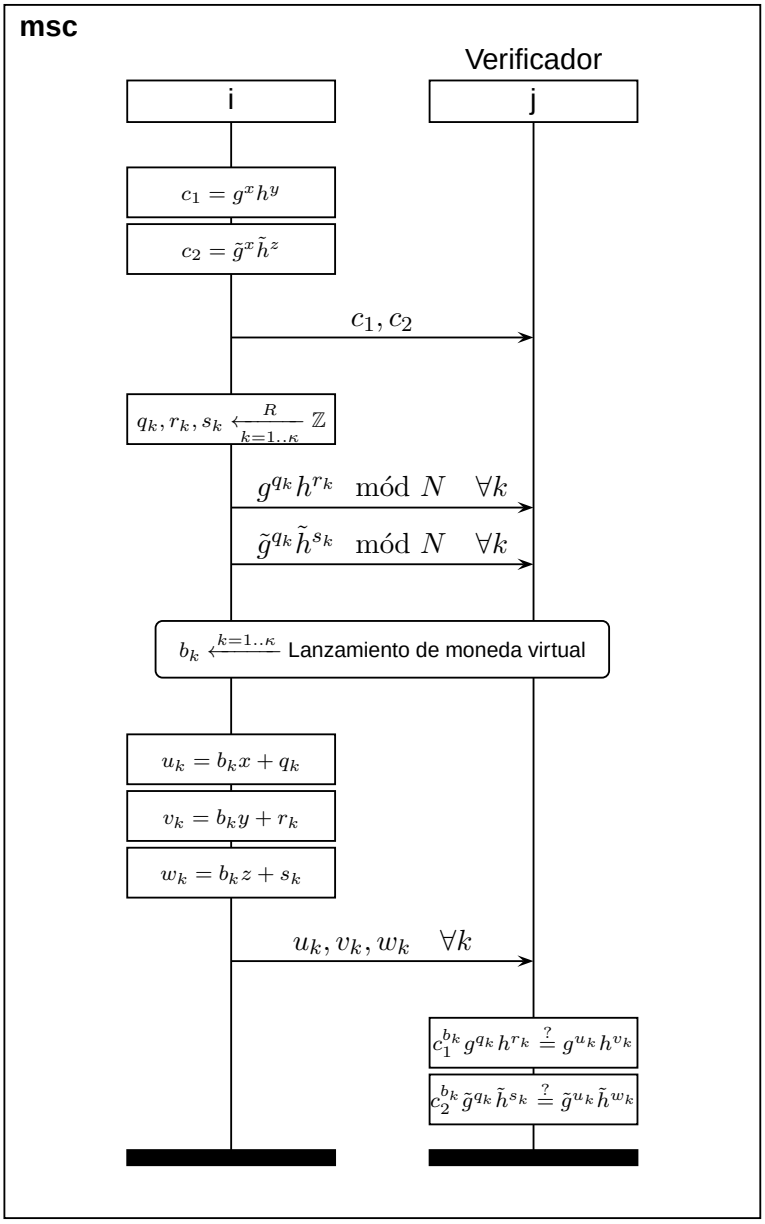


Figura 2.3: Prueba de conocimiento de igualdad de logaritmos

[7] Se trata de compromisos de Damgård – Fujisaki

Acto seguido, ambos participantes realizan un lanzamiento de moneda virtual obteniendo así un bit aleatorio b que se usará para calcular los valores $u = bx + q$, $v = by + r$ y $w = bz + s$.

Finalmente, se envían estos tres valores u, v y w al verificador para que compruebe que se cumplen las relaciones $c_1^b g^q h^r = g^u h^v$ y $c_2^b \tilde{g}^q \tilde{h}^s = \tilde{g}^u \tilde{h}^w$.

Si se cumplen ambas relaciones durante las κ iteraciones se considera superada la prueba de conocimiento.

2.4. Firmas digitales

Es preciso finalizar este apartado de primitivas criptográficas con una breve introducción a los esquemas de firma digital.

La firma digital es una primitiva criptográfica que, aplicada sobre un mensaje M, debe asegurar las siguientes propiedades:

Autenticidad El receptor del mensaje puede estar seguro de quién es el emisor.

No-Repudio El emisor del mensaje no puede negar su autoría.

Integridad Un mensaje firmado no puede ser modificado por terceras partes sin que el receptor se dé cuenta de ello.

Algunos esquemas de firma digital se basan en una Infraestructura de Clave Pública[8] a través de la cual todo usuario puede obtener de manera fiable la clave pública e_k de cualquier otro usuario. La PKI proporciona, además, los algoritmos de cifrado y descifrado necesarios ($E(\cdot, e_k)$ y $D(\cdot, d_k)$ respectivamente). Por último, cada usuario dispone de una clave secreta d_k tal que: $E(D(m, d_k), e_k) = m$.

Así, algunos esquemas elementales de firma digital basados en una PKI aprovechan las funciones E y D para genenerar una firma del tipo $\sigma_i(m) = D(m, d_i)$, donde d_i es la clave privada del usuario i.

Si un usuario j quisiera verificar que el emisor es i y que el mensaje no ha sido manipulado debería comprobar que $m = E(\sigma_i(m), e_i)$. Esta verificación puede ser llevada a cabo por cualquier usuario dado que tanto las funciones E y D como la clave e_i son públicas.

Además, el usuario j puede demostrar ante cualquier otro usuario que ha recibido el mensaje m simplemente revelando el par $(m, \sigma_i(m))$ para que dicho usuario realice la misma comprobación (que no depende de j).

[8] En adelante PKI, del Inglés *Public Key Infraestructure*

Como puede verse, la seguridad de un esquema de firma depende fuertemente de que para todo usuario j (con $j \neq i$) sea imposible generar un par $(m, \sigma_i(m))$ válido, que es una hipótesis algo más fuerte que la hipótesis estándar de los esquemas de cifrado de clave pública[9].

Con el fin de mejorar su eficiencia, es frecuente combinar los esquemas de firma con una función resumen[10] $H(\cdot)$. Estas funciones toman una entrada m de tamaño arbitrario y devuelven un elemento de un conjunto finito (generalmente un invervalo de $\mathbb{Z}$).

Así, para firmar un mensaje m el usuario i calculará el valor $\sigma_i(m) = D(H(m), d_i)$ y para verificar la firma el usuario j comprobará que $H(m) = E(\sigma_i(m), e_i)$.

Esta modificación del protocolo original depende de las hipótesis antes mencionadas y, además, de lo resistente a colisiones que sea la función resumen. En particular, para garantizar la integridad y el no-repudio del mensaje firmado es imprescindible que dado un valor $H(m)$ sea virtualmente imposible encontrar un valor m' tal que $H(m) = H(m')$.

2.4.1. Firmas de umbral

En ocasiones se precisa que un determinado grupo de usuarios firmen un documento de manera conjunta según establezca una determinada estructura de acceso Γ (ver sección 2.2.1).

Así, dado un conjunto C de usuarios, se requiere que cualquier subconjunto A tal que $A \in \Gamma$ pueda generar una firma válida para un mensaje m dado mientras que cualquier subconjunto no autorizado sea totalmente incapaz de ello.

Con frecuencia se exige, además, que el tamaño de la firma obtenida no dependa del tamaño de A ni del tamaño de C. También es usual imponer que no se pueda identificar el subconjunto concreto de usuarios que ha generado la firma (proporcionando así cierto anonimato) si bien en ocasiones es útil precisamente lo contrario.

Por último, la estructura de acceso más frecuente para las firmas descritas en los párrafos precedentes es la estructura de umbral (ver sección 2.2.1). Las firmas que cumplen lo anterior para dicha estructura de acceso se llaman firmas de umbral y son de gran utilidad en numerosos protocolos criptográficos.

[9] Dicha hipótesis establece que, dado $E(m, e_i)$ es imposible recuperar el valor de m pero esto no implica que no se pueda encontrar un valor m', probablemente sin sentido, tal que $E(m, e_i) = E(m', e_i)$

[10] En la literatura criptográfica es frecuente referirse a las funciones resumen por su denominación inglesa *Hash fuctions*.

2.4.2. RSA

Antes de dar por finalizada esta introducción a las firmas digitales es preciso dar un ejemplo de firma digital de uso habitual, la firma RSA.

Los protocolos de firma y cifrado RSA fueron descritos por Ron Rivest, Adi Shamir y Len Adleman en [RSA78] y aún hoy constituyen la base de la mayoría de protocolos de clave pública.

Cifrado RSA

Cifrar un mensaje con un esquema RSA consiste, básicamente, en realizar la exponenciación modular:

$$E(m, e) = m^e \mod N$$

donde m es el mensaje a firmar, e es la clave pública y N es un módulo RSA, es decir, $N = pq$ con p y q primos y de gran tamaño (actualmente se usan primos de 1024 bits).

Para descifrar tan sólo es necesario disponer de $d \equiv e^{-1} \mod \phi(N)$, donde $\phi(N) = (p-1)(q-1)$, y realizar la siguiente exponenciación modular:

$$D(E(m, e), d) = E(m, e)^d \mod N = m^{ed} \mod N = m$$

Nótese que es necesario que $\gcd(e, \phi(N)) = 1$ para que el proceso de cifrado sea invertible, y que para obtener la clave privada d basta con disponer de $\phi(N)$[11], que a su vez se puede calcular con facilidad a partir de p y q. La seguridad del cifrado RSA depende, por lo tanto, de la dificultad de la factorización de enteros de gran tamaño.

Firma RSA

El proceso de firma es totalmente análogo al de cifrado pero intercambiando el uso de la clave pública y la clave privada. Así, para generar una firma RSA del mensaje M se debe calcular:

$$Sign(m) = D(m, d) = m^d \mod N$$

y enviar el par $(m, Sign(m))$ al receptor para que pueda comprobar que, en efecto, $Sign(m)^e \mod N = m$.

[11] La identidad de Bezout establece que existen a, b tales que $ae + b\phi(N) = \gcd(e, \phi(N))$ y el algoritmo extendido de Euclides nos permite calcular a y b eficientemente a partir de e y $\phi(N)$ obteniendo, $a \equiv e^{-1} \mod \phi(N)$

Firma RSA de umbral

El esquema de firma RSA puede combinarse con el esquema de compartición de secretos de Shamir para obtener una firma de umbral como las descritas en la sección 2.4.1.

En particular, haciendo uso de la técnica de reconstrucción en el exponente (ver sección 2.2.2) podemos realizar la exponenciación modular de la firma RSA con un exponente compartido mediante un polinomio aleatorio $p(x)$ tal que $p(0) = d$:

$$D(m, d) = m^d \mod N = \prod_{i \in A} \sigma_i^{\lambda_i^S(0)} \mod N$$

donde $\sigma_i = m^{p(i)} \mod N$ son las *firmas parciales* y $\lambda_i^A(x)$ son los coeficientes de Lagrange correspondientes a los participantes que están generando la firma ($i \in A \in \Gamma$).

Así pues, cada participante del conjunto autorizado A genera su firma parcial σ_i y la envía al encargado de reconstruir la firma, que deberá combinarlas para obtener $D(m, d)$.

Nótese que en ningún momento se reconstruye explícitamente la clave secreta d y, por lo tanto, es imposible falsificar una firma $D(m', d)$ a partir de las firmas parciales del mensaje m. Además es necesario que ningún usuario conozca p, q o $\phi(N)$ ya que de lo contrario podría calcular d y firmar por su cuenta.

También es de vital importancia observar que los elementos en el exponente son de $\mathbb{Z}_{\phi(N)}$ y que, al ser $\phi(N)$ desconocido para todos los participantes, será necesario usar técnicas especiales (análogas a las que se explican en la sección 2.2.3) para calcular los coeficientes de Lagrange.

Capítulo 3

Protocolo robusto para firmas RSA distribuidas

Una vez vistos los antecedentes (sección 0.2) y explicadas las principales herramientas necesarias (capítulo 2) ha llegado la hora de entrar de lleno en el protocolo distribuido y robusto de firma RSA.

El protocolo descrito a continuación ha sido adaptado a los requisitos específicos de los procesos de voto eletrónico (sección 1). En particular se han usado dos hipótesis no estándar, a saber, que el número de participantes sería del orden de la decena y que habría más de un 50 % de usuarios honestos. Así mismo se ha requerido que tanto la generación de parámetros RSA como el proceso de firma sean totalmente distribuidos y que el protocolo sea robusto, es decir, que se necesite la colaboración de múltiples usuarios para llevarlo a cabo y que ninguna minoría maliciosa pueda impedir que el protocolo termine exitosamente.

Como puede observarse en la figura 3.1, el procolo se divide en dos partes bien diferenciadas, la **Generación distribuida de los parámetros RSA** (sección 3.1) y el proceso de **Firma RSA distribuida** (sección 3.2). Ambas partes podrían ser substituidas modularmente por otras que realizasen la misma función. Es decir podríamos usar otro procolo de generación de parámetros RSA[1] siempre que éste proporcione a los usuarios un módulo RSA N, una clave pública e y los fragmentos d_i correspondientes a la estructura de acceso usada en la firma distribuida. Así mismo podrían usarse los parámetros generados en la primera parte del protocolo para otros usos menos habituales como, por ejemplo, un esquema en el que se requiera la colaboración de un grupo de personas para descifrar un mensaje cifrado con la clave pública RSA (N, e).

A continuación se detallan los pasos que sigue cada protocolo y las modificaciones realizadas respecto a lo descrito en los artículos mencionados en la sección 0.2.

[1] Podríamos, por ejemplo, confiar en una tercera parte de confianza, ganando en eficiencia pero perdiendo la naturaleza distribuida del protocolo a nivel global.

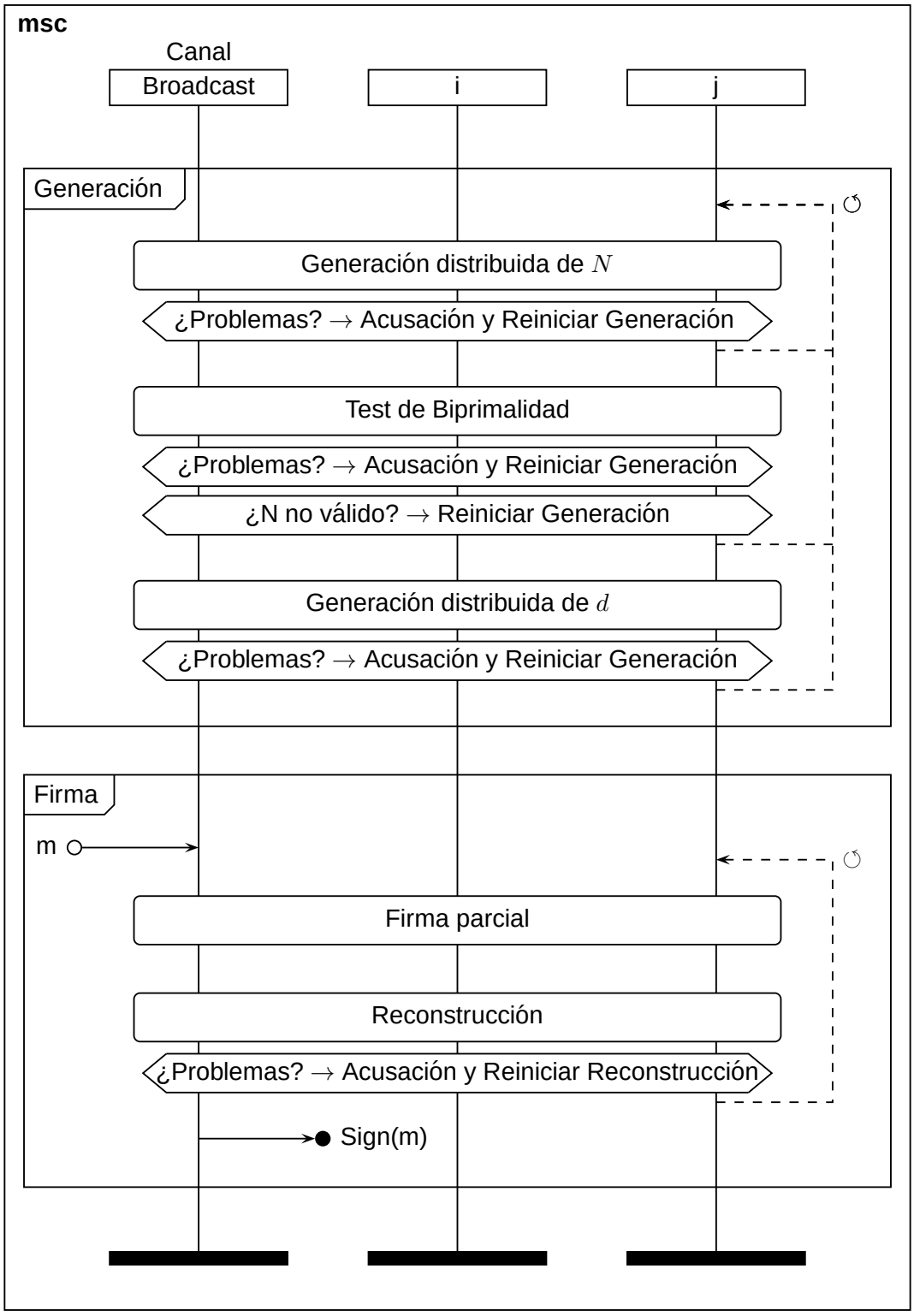


Figura 3.1: Esquema general del protocolo

3.1. Generación distribuida de parámetros RSA

La generación de parámetros RSA se basa en el esquema propuesto en [FMY98], que a su vez se basa en el esquema de [BF97]. Ambas propuestas usan una estructura en tres partes, a saber, generación y multiplicación distribuida de N, test de biprimalidad sobre N y generación de los fragmentos de clave secreta d_i.

3.1.1. Generación distribuida de N

Para obtener los parámetros necesarios para realizar una firma RSA no distribuida es necesario disponer de información privilegiada. Concretamente, a partir del par de primos p, q se pueden obtener $N = pq$ y $d \equiv e^{-1} \mod (p-1)(q-1)$. Sin embargo, en la versión distribuida no disponemos explícitamente de dichos valores sino de fragmentos de dichos valores. Así, al principio del protocolo de generación de parámetros RSA, cada participante generará los valores p_i y q_i que le permitirán encontrar los parámetros (N, e, d_i) con la colaboración del resto de participantes.

De manera implícita estaremos trabajando con valores de p y q que cumplen las relaciones:

$$p = \sum_{i=1}^{n} p_i \qquad y \qquad q = \sum_{i=1}^{n} q_i$$

sin embargo dichos valores no se reconstruirán explícitamente en ningún punto del protocolo (a menos que se detecten irregularidades, momento en el cual quizá sea necesario calcular dichos valores para expulsar a los usuarios maliciosos).

Al generar p y q de manera aditiva en base a unos fragmentos aleatorios escogidos de manera independiente es relativamente difícil conseguir que ambos sean primos. En concreto, si tenemos en cuentra que en promedio es necesario hacer h pruebas antes de obtener un número primo de h bits, es obvio que tardaremos aproximadamente h^2 intentos en encontrar un módulo RSA N que sea producto de dos números primos de h bits. La idea es calcular N y comprobar su *biprimalidad* hasta obtener un módulo válido. Para los parámetros que se manejan actualmente en el ámbito de la votación electrónica podemos esperar que este proceso se repita $1{,}000{,}000$ de veces.

Por motivos que quedarán claros en la sección 3.1.2 nos interesa que tanto p como q sean congruentes con $3 \mod 4$. A tal efecto el participante 1 deberá escoger sus fragmentos de tal manera que $p_1 \equiv 3 \mod 4$ y $q_1 \equiv 3 \mod 4$ mientras que el resto de participantes deberían escoger sus respectivos p_i y q_i de tal manera que $p_i \equiv 0 \mod 4$ y $q_i \equiv 0 \mod 4$. Este requerimiento no afecta de manera significativa a lo explicado en el párrafo anterior.

Esta parte del protocolo está representada en las figuras 3.2 y 3.3.

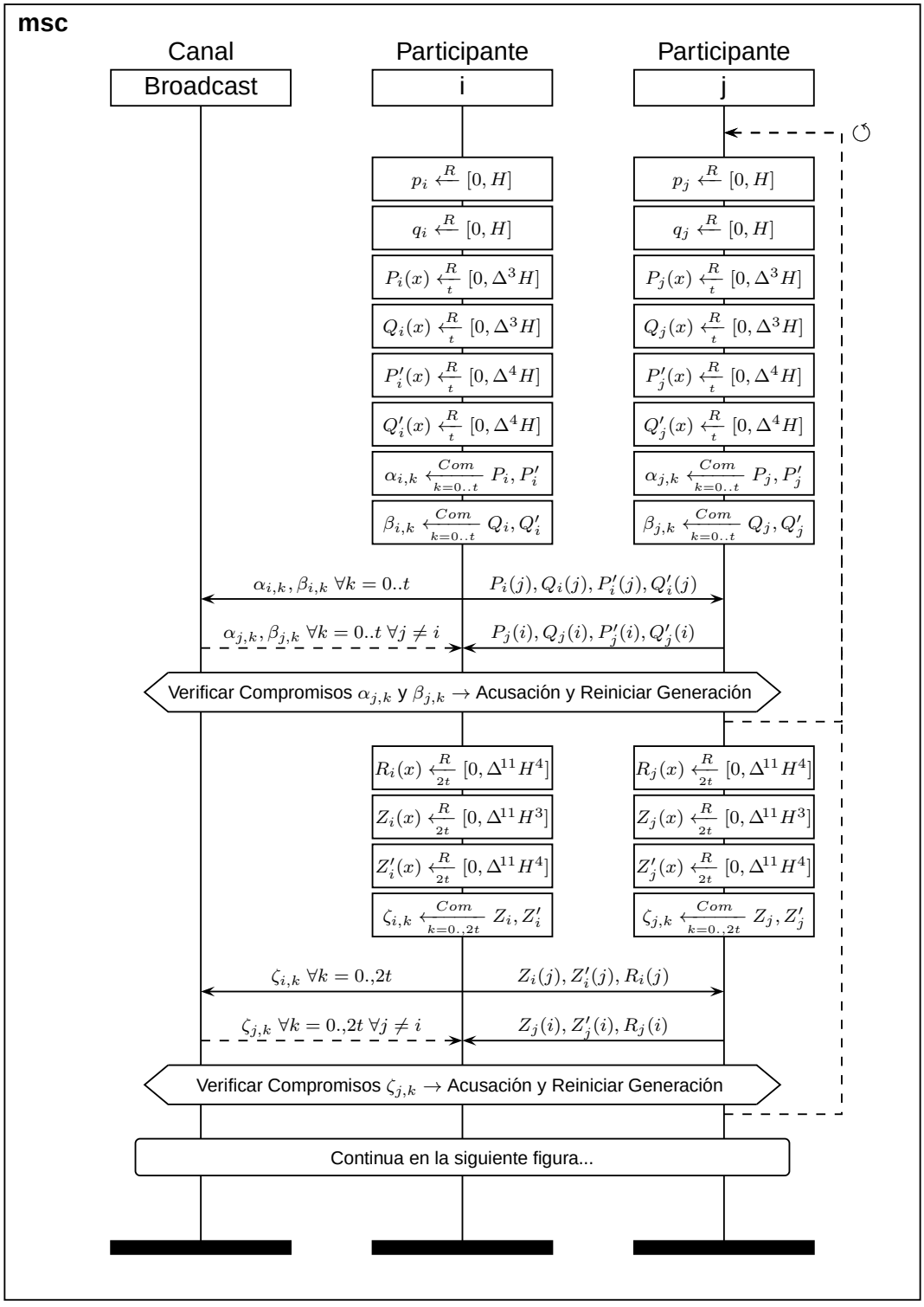


Figura 3.2: Generación distribuida de N (1 de 2)

En la primera de ellas pueden verse cómo cada participante elige sus dos fragmentos p_i y q_i de manera aleatoria y uniforme en el intervalo $[0, H]$, donde $2\log_2 H$ es el número de bits que tiene que tener N.

A continuación genera dos polinomios $P_i(x)$, $Q_i(x)$ de grado t cuyos coeficientes se eligen de manera aleatoria y uniforme en el intervalo $[0, \Delta^3 H]$ de manera sean múltiplos de $\Delta(= n!)$ y tales que $P_i(0) = \Delta^2 p_i$ y $Q_i(0) = \Delta^2 q_i$ como ya se explicó en la sección 2.2.

Así mismo se generan unos polinomios auxiliares $P'_i(x)$ y $Q'_i(x)$, sin ninguna restricción en lo que al término independiente se refiere, que servirán para generar los compromisos $\alpha_{i,k}$ y $\beta_{i,k}$ respectivamente tal y como se explicó en la sección 2.3.

El siguiente paso consiste en generar un polinomio aleatorio $Z_i(x)$ de grado $2t$ tal que $Z_i(0) = 0$ y su correspondiente polinomio auxiliar $Z'_i(x)$ para poder obtener los compromisos $\zeta_{i,k}$.

Finalmente se genera un último polinomio aleatorio $R_i(x)$, también de grado $2t$, que servirá para aleatorizar los compromisos.

Conforme se generan estos polinomios se envía, a cada usuario, su correspondiente fragmento. Así, al final del protocolo el jugador $i-$ésimo dispondrá de los valores $P_j(i), P'_j(i), Q_j(i), Q'_j(i), Z_j(i), Z'_j(i)$ y $R_j(i)$ para todo valor de j entre 1 y n y, además, podrá obtener del canal de broadcast los compromisos $\alpha_{j,k}, \beta_{j,k}$ y $\zeta j, k$ para todo valor de j entre 1 y n y para todo valor de k entre 0 y t (o entre 0 y $2t$ en el caso de $\zeta_{j,k}$).

Con toda esa información podrá asegurarse de que, en efecto, todos los valores se corresponden a evaluaciones de polinomios del grado que toca. En caso contrario se procedería a denunciar públicamente al usuario que haya proporcionado datos erróneos y se harán públicos los mismos para que cada cual pueda comprobar quién miente (si el acusado o el acusador) y expulsar de común acuerdo al usuario malicioso[2].

Una vez recibidos y comprobados todos esos fragmentos, estos se usarán para generar compromisos del polinomio $N_i(x) = P(x)Q_i(x) + Z_i(x)$, donde $P(x) = \sum_k P_i(x)$. También se publica el valor $N(j) = (\sum P_k(j))\,(\sum Q_k(j)) + \sum Z_k(j)$ y se generan los compromisos $V_{j,k}$ que se usarán para comprobar dicho valor.

A continuación se realizarán pruebas de conocimiento (explicadas en la sección 2.3) para demostrar igualdad de logaritmos entre:

$$g^{Q_j(i)} h^{Q'_j(i)} \qquad y \qquad \left(g^{P(i)} h^{P'(i)}\right)^{Q_j(i)} h^{R_j(i)}$$

Por último, cada participante interpola el valor de $N(0)$ y lo divide por Δ^4 para obtener el módulo RSA N.

[2]Descartando tanto a acusado como a acusador aún tenemos una mayoría de usuarios honestos de manera que una simple votación debería ser suficiente para determinar quien miente. Además, cualquiera que apoye al usuario malicioso en dicha votación puede ser considerado, así mismo, malicioso.

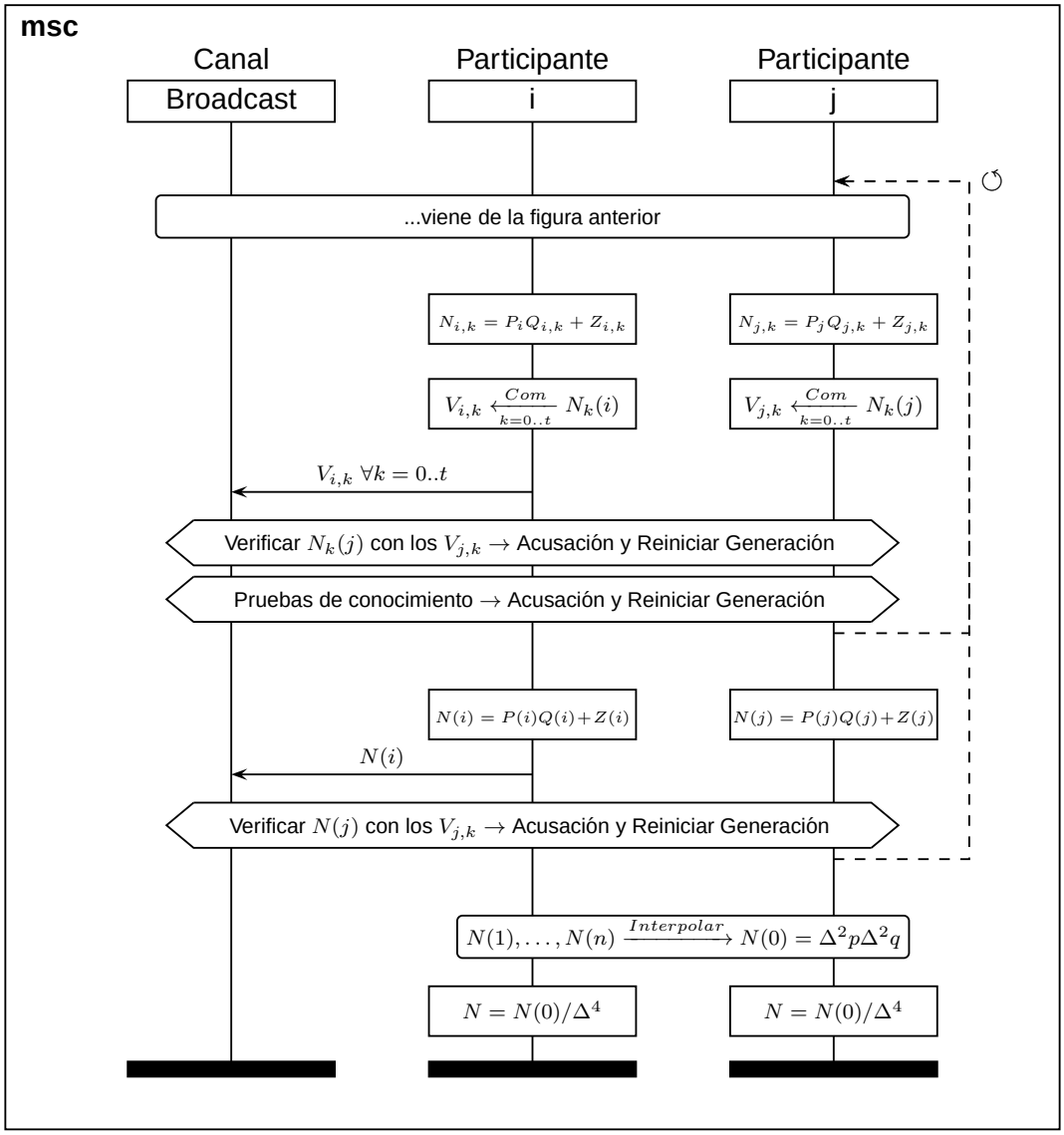


Figura 3.3: Generación distribuida de N (2 de 2)

En este punto del protocolo hemos obtenido el producto N de dos valores p y q sin que ningún usuario pueda, de manera unilateral, obtener ninguno de los dos valores.

Así mismo, y dado que todos los participantes (y en particular los participantes honestos) han aportado sus fragmentos p_i, q_i, podemos estar seguros de que éstos se hallan en el rango adecuado.

Ahora queda por comprobar que, efectivamente, tanto p como q son números primos.

3.1.2. Test de biprimalidad

Para poder utilizar el módulo N en una firma RSA es necesario que éste sea producto de dos primos p y q. Por el método de construcción utilizado (ver sección 3.1.1) sabemos que N es, en efecto, producto de dos enteros p y q. Pero dichos números se han generado de manera aleatoria con la colaboración de todos los participantes y, además, no disponemos de los mismos de manera explícita así que, a priori, no tenemos ninguna garantía de que sean primos.

Para asegurarnos de que N es un módulo válido se necesita un test de primalidad. La propuesta que se presenta a continuación se basa en el test de biprimalidad de Boneh y Franklin [BF97] que más tarde harían robusto Frankel et al. en [FMY98].

Se trata, de hecho, de un test de biprimalidad que comprueba al mismo tiempo la primalidad de p y q. Dicho test está basado en el test de primalidad de Fermat (ver, por ejemplo, [DK02]), que, a su vez, tiene su origen en el teorema de Euler, que afirma que para todo N y para todo g tal que $\gcd(g, N) = 1$ se da la siguiente congruencia:

$$g^{\phi(N)} \equiv 1 \mod N$$

El test tan sólo sirve para valores de N que sean Enteros de Blum, es decir, tales que $N = pq$ con $p \equiv 3 \mod 4$ y $q \equiv 3 \mod 4$. Esto ya se ha tenido en cuenta durante la generación distribuida de N (sección 3.1.1). Se requiere además que el Símbolo de Jacobi de g cumpla: $\left(\frac{g}{N}\right) = 1$.

Una vez se cumplen ambas condiciones ya se puede calcular de manera distribuida el valor $g^{\phi(N)/4}$ que, si p y q son primos, equivale a:

$$\begin{aligned} g^{\phi(N)/4} &= \left(g^{(p-1)/2}\right)^{(q-1)/2} \equiv \left(\frac{g}{p}\right)^{(q-1)/2} = \left(\frac{g}{p}\right) \mod p \\ g^{\phi(N)/4} &= \left(g^{(q-1)/2}\right)^{(p-1)/2} \equiv \left(\frac{g}{q}\right)^{(p-1)/2} = \left(\frac{g}{q}\right) \mod q \end{aligned}$$

Como $\left(\frac{g}{N}\right) = 1$ resulta que $\left(\frac{g}{p}\right) = \left(\frac{g}{q}\right)$, que, junto a las congruencias que se acaban de ver, implica que $g^{\phi(N)/4} \equiv \pm 1 \mod N$.

En caso de que o p o q (o ambos) no sean primos el test puede dar falsos positivos con probabilidad $1/2$. Es decir, si el test da positivo la probabilidad de que N sea un módulo RSA válido es mayor o igual al $50\,\%$ mientras que si el test da negativo sabremos que N no es un módulo RSA válido[3].

Para mejorar la fiabilidad del test podemos repetirlo κ veces de manera independiente para obtener una probabilidad de falso positivo menor o igual a $1/2^{\kappa}$.

[3] La demostración de este hecho y otros detalles interesantes respecto a este test de biprimalidad pueden hallarse en [BF97].

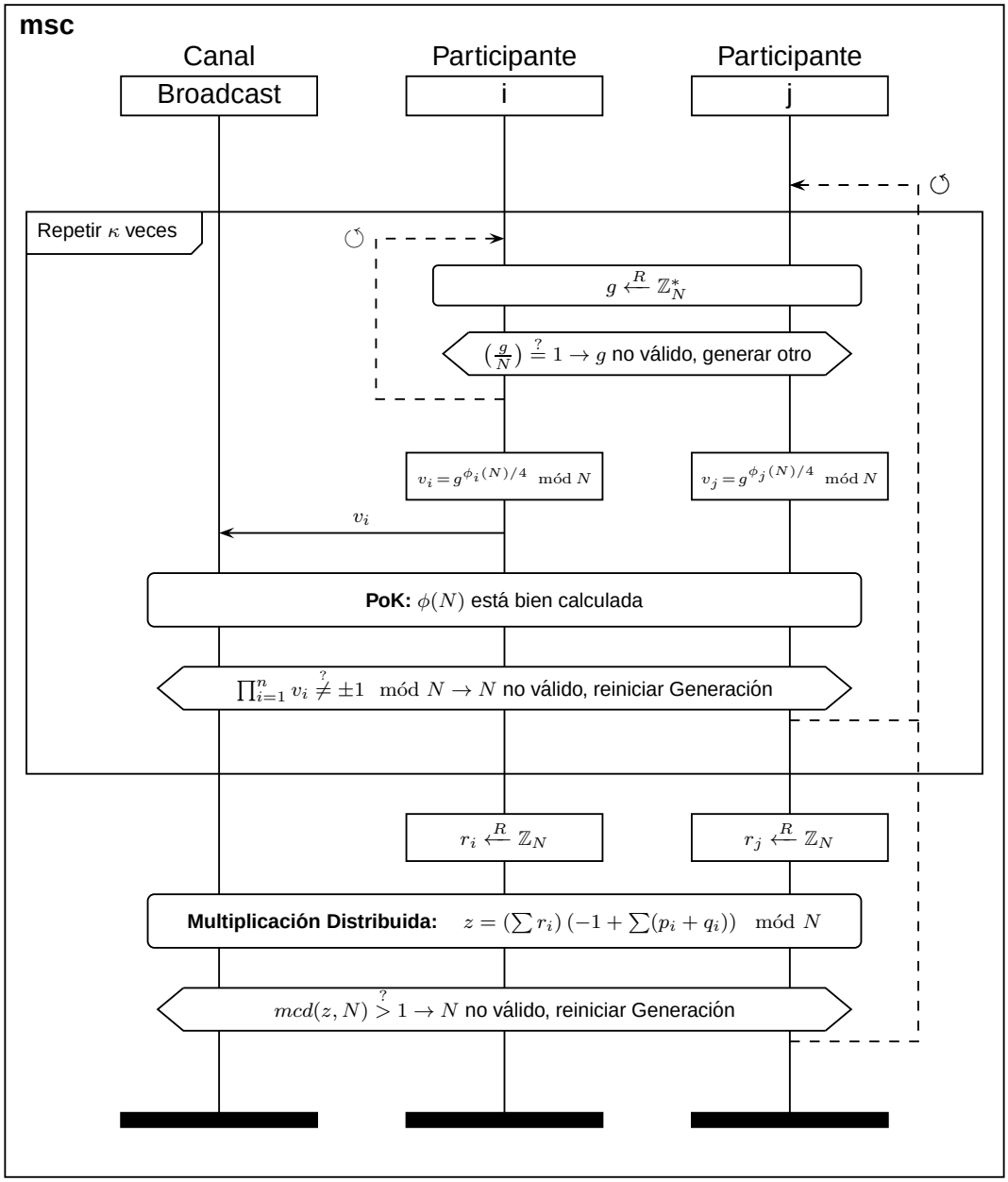


Figura 3.4: Test de Biprimalidad de N

Se recomienda, además, añadir un último paso que elimina aquellos números que, como sucede con los enteros de Carmichael en el test de Fermat estándar, superan siempre el test pese a no ser válidos.

Dichos números son de la forma $N = pq$ con $p = r_1^{d_1}$ y $q = r_2^{d_2}$, siendo r_1 y r_2 números primos y cumpliendo que $d_1 > 1$ y $q \equiv 1 \mod r_1^{d_1 - 1}$. Los valores de N que cumplen dichas condiciones siempre superan las primeras iteraciones del test pero son descartados con esta última prueba.

Lamentablemente, este último paso descarta también algunos módulos válidos (falsos negativos) pero la probabilidad de obtener dichos módulos es despreciable y, además, siempre es preferible obtener falsos negativos ya que estos ralentizan el protocolo pero no afectan a la seguridad mientras que los falsos positivos podrían comprometer la seguridad de la firma.

En la figura 3.4 pueden verse resumidos los pasos del test de biprimalidad.

Para empezar es necesario elegir aleatoriamente un entero g con símbolo de Jacobi $\left(\frac{g}{N}\right) = 1$. Esto se puede hacer usando un mismo generador pseudoaleatorio para que todos los participantes inicialicen con la misma semilla.

A continuación cada participante calcula su fragmento de $\phi(N) = \sum \phi_i(N)$:

$$\begin{aligned} \phi_1(N) &= N - p_1 - q_1 + 1 \\ \phi_i(N) &= -p_i - q_i \end{aligned}$$

y publica el valor $v_i = g^{\phi_i(N)/4} \mod N$.

Para verificar que, en efecto, los valores v_i están bien calculados se realizan unas pruebas de conocimiento[4] de igualdad de logaritmo discreto como las descritas en la sección 2.3.4. En particular se comprueba que los compromisos de p_i y q_i publicados durante la generación de N sean coherentes con el valor de v_i publicado en este paso.

Por último se calcula el valor $\prod_{i=1}^{n} v_i \mod N$. En caso de que este valor sea igual a 1 o -1 declaramos que N es válido con probabilidad $1/2$. En caso de que sea diferente de ± 1 descartamos N y reiniciamos el protocolo de generación de parámetros RSA.

Como ya se ha explicado antes, este test se repite κ veces para conseguir una fiabilidad de, al menos, $2^{-\kappa}$.

Por último, si N supera κ rondas de este test, se realiza un último paso que consiste en generar un valor aleatorio r de manera implícita ($r = \sum r_i$) y calcular el producto $z = \left(\sum r_i\right)\left(-1 + \sum(p_i + q_i)\right) \mod N$ usando el mismo protocolo que el que se ha usado en la sección 3.1.1 para calcular N.

Con el valor z se calcula el máximo común divisor de z y N. Si ambos son coprimos declaramos que N es un módulo RSA válido. En caso contrario descartamos N y reiniciamos el proceso de generación de parámetros RSA.

[4] En adelante *PoK* del Inglés *Proof of Knowledge*

Escenario optimista

Las pruebas de conocimiento que hay que realizar en el proceso de generación de N y en el test de biprimalidad resultan muy costosas. Esto es así dado que no puede usarse la heurística de Fiat–Shamir [FS86] para evitar la interactividad de las mismas.

Por todo ello es interesante evitar dichos pasos siempre que sea posible.

Usando un generador pseudoaleatorio fiable es posible implementar una variante del protocolo en el que estas pruebas de conocimiento tan sólo se realizan en caso de que se detecte alguna irregularidad.

La idea básica es realizar el protocolo sin pruebas de conocimiento hasta el test de biprimalidad. En este punto, si el test da negativo se sabe que los fragmentos p_i y q_i no valen para nada y, por lo tanto, se pueden publicar para que cada cual compruebe que, en efecto, o p o q (o ambos) son compuestos. Si ambos fuesen primos algún participante habría mentido y sería necesario publicar la semilla del generador pseudoaleatorio utilizada por cada participante para que todos los demás comprueben que la información publicada por ese participante fue correctamente calculada.

Si llegados al test de biprimalidad resulta que N es aceptado entonces se debe repetir, usando los mismos valores p_i y q_i, tanto la multiplicación distribuida de p y q como el test de biprimalidad realizando, ahora sí, las pruebas de conocimiento. Si en esta segunda ejecución del protocolo se detectase alguna irregularidad (el nuevo N no concuerda con el anterior o es declarado inválido en el test de biprimalidad) deberíamos revelar los fragmentos de p y q y las semillas aleatorias usadas para que todos los participantes comprueben quién ha mentido.

Con esta versión del protocolo, las pruebas de conocimiento, que son el cuello de botella del protocolo, tan sólo se realizan en una ocasión (cuando N es un módulo RSA válido) en vez de realizarse cada vez que se genera un candidato a módulo RSA (del orden de 10^6 veces para primos de 1024 bits). En las otras ocasiones, aquellas en las que el módulo generado no es válido y que por lo tanto no es necesario mantener los parámetros privados en secreto, cada jugador puede comprobar que el resto ha calculado correctamente los valores que ha ido publicando de manera asíncrona y con un coste mucho menor del que suponen las pruebas de conocimiento.

Como ya se ha dicho, también es posible que se deban ejecutar las pruebas de conocimiento si algún jugador miente y consigue que un valor de N no válido pase el test de biprimalidad en la primera ejecución, pero dado que cada vez que esto pase eliminaremos, al menos, a un participante malicioso, esta situación tan sólo se dará unas $n/2$ veces.

3.1.3. Generación distribuida de d

El último paso que es necesario realizar antes de poder firmar mensajes usando el módulo RSA que tanto ha costado encontrar es el proceso de generación de las claves pública y privada (e y d respectivamente).

La clave pública e servirá para verificar la firma. De hecho, para verificar que una determinada firma $\sigma = Sign(m)$ es correcta basta con realizar la exponenciación modular:

$$m' = \sigma^e \mod N$$

y comprobar que $m' = m$.

En general tenemos cierta libertad a la hora de escoger e y la única condición que impondremos será que sea un número primo suficientemente grande (de más de 16 bits).

En realidad también es necesario que e y $\phi(N) = (p-1)(q-1)$ sean coprimos, pero resulta difícil comprobar tal extremo dado que no disponemos explícitamente del valor de $\phi(N)$[5]. De hecho es imprescindible mantener $\phi(N)$ en secreto dado que, de lo contrario, cualquiera podría encontrar la clave privada d y firmar mensajes de manera unilateral.

Una vez fijada una clave pública e es necesario generar la clave pivada d que se corresponde con esta. Concretamente buscamos una d tal que:

$$d \equiv e^{-1} \mod \phi(N)$$

Para ello se seguirá el protocolo propuesto en [BF97] y mejorado en [FMY98].

Dicho protocolo calcula la clave pública d a partir de los valores p_i y q_i que los participantes han usado en la generación de N además del propio valor de N (que ahora es ya público) y del valor de e.

Lo primero que se debe hacer es calcular de manera distribuida $\psi = r \cdot \phi(N) \mod e$ usando el protocolo de multiplicación distribuida descrito en la sección 3.1.1. Esto puede llevarse a cabo dado que los participantes disponen de fragmentos aditivos de r y $\phi(N)$.

En dicha multiplicación distribuida se comprobará, mediante unas pruebas de conocimiento de igualdad de logaritmo discreto (sección 2.3.4) que el valor de $\phi(N)$ usado en este paso es coherente con los compromisos publicados por cada participante durante la generación de N.

[5] En su lugar, lo que se puede comprobar con facilidad es que $\psi = r \cdot \phi(N)$ sea invertible módulo e donde r es un valor aleatorio que se genera de manera distribuida (cada participante elige un r_i y se trabaja con el valor implícito $r = \sum r_i$). De esta manera es posible que ψ y e no sean coprimos por culpa de r y, en consecuencia, es razonable probar diversos valores de r antes de descartar una clave pública (descartar $\phi(N)$ no es una opción dado que implicaría reiniciar el costoso proceso de generación de N).

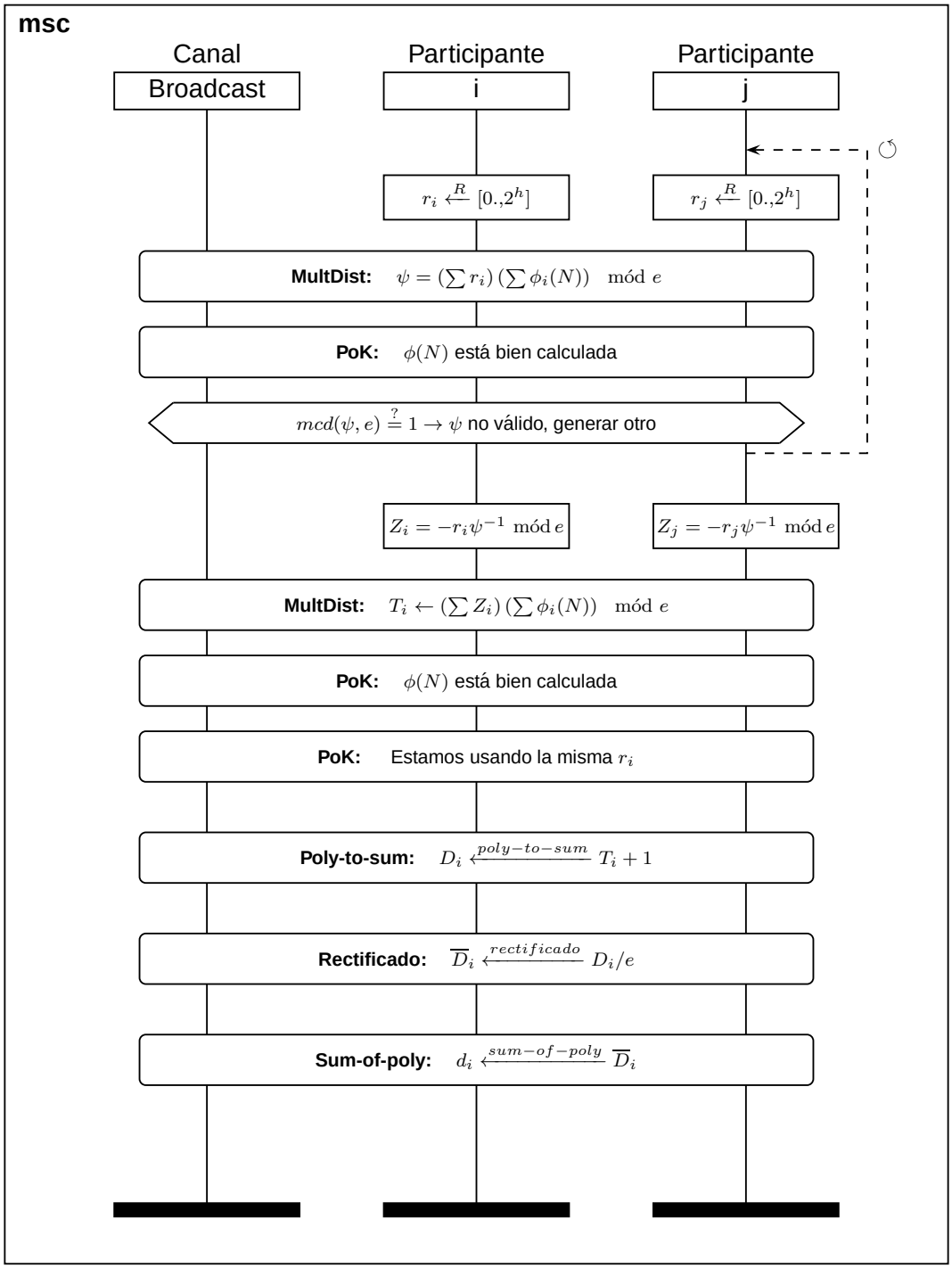


Figura 3.5: Generación de los fragmentos de d

A continuación se reconstruirá y se hará público el valor de ψ para que cada participante pueda calcular $Z_i = -r_i \cdot \psi^{-1} \mod e$. En caso de que ψ no sea invertible módulo e es necesario reiniciar el protocolo de generación de d con nuevos valores de r_i. Si después de probar varios valores de r no se ha obtenido un ψ invertible es probable que $\phi(N)$ sea múltiplo de e y será necesario buscar un nuevo valor de e antes de proseguir.

Con los valores Z_i los participantes realizan una segunda multiplicación distribuida para obtener fragmentos de $T = \sum Z_i \cdot \phi(N) \mod e$. En dicha multiplicación distribuida se realizan dos pruebas de conocimiento para verificar que tanto los fragmentos de r como los fragmentos de $\phi(N)$ usados son coherentes con los usados en pasos anteriores.

Nótese que $T + 1 \equiv 0 \mod e$ y $T + 1 \equiv 1 \mod \phi(N)$ de manera que $T + 1 = ee^{-1} \mod \phi(N)$ y tan sólo tenemos que realizar una división para encontrar el valor de la clave privada que buscamos.

Los fragmentos $T_i + 1$ ocultan el secreto $T + 1$ mediante un esquema polinómico de Shamir, que es cómo deseamos obtener la clave privada d. Sin embargo es mucho más sencillo realizar la división anteriormente citada si los fragmentos son aditivos. Para ello se usa la transformación *Poly-to-sum* explicada en la sección 2.2.2 para obtener fragmentos aditivos D_i de $d \cdot e \mod \phi(N)$, se divide cada uno de dichos fragmentos por e sin tener en cuenta el resto de dicha división. Para corregir los errores derivados de no tener en cuenta dicho resto se realiza un proceso de rectificado que consiste en firmar un mensaje m carente de siginificado de la siguiente manera:

Cada participante calcula y publica $\sigma_i = m^{D_i/e} \mod N$. Todos los participantes calculan $\sigma = \prod_i \sigma_i \mod N$. Si no se hubiesen cometido errores de redondeo sucedería que $m = \sigma^e \mod N$ ya que $\sigma = m^{\sum D_i/e} \mod N = m^{de/e} \mod N$.

En caso de que σ no sea una firma válida de m el error cometido será pequeño (menor que n) y podrá corregirse por ensayo y error. Para ello basta con multiplicar σ por m o por m^{-1} hasta obtener una firma válida. Sea ε el número de veces que se ha multiplicado σ por m en su rectificado, entonces basta con que uno de los participantes (por ejemplo el participante número 1) sume dicho valor (que puede ser negativo) a su fragmento de la clave privada D_i.

Por último debe restablecerse el esquema de Shamir realizando un *Sum-of-poly* como se ha explicado en la sección 2.2.2. En este punto del protocolo los participantes disponen de los parámetros públicos N y e y de fragmentos polinómicos d_i de la clave privada d.

En términos de eficiencia ésta es una parte poco siginificativa del protocolo. Hay que tener en cuenta que, pese a requerir dos multiplicaciones distribuidas y tres pruebas de conocimiento (que son muy costosas y altamente interactivas), la generacion de la clave privada d tan sólo se ejecuta una vez mientras que en la generación de N se esperan varios miles de iteraciones.

3.2. Firma distribuida

El objetivo fundamental de este proyecto es la consecución de un protocolo de firma distribuida. A tal efecto, se han generado los parámetros RSA necesarios para la realización de la firma RSA distribuida y podemos asumir en este punto que los n participantes disponen de fragmentos de la clave privada d que permiten reconstruir ésta en virtud de alguna estrucutra de acceso (típicamente una estructura de acceso de umbral t-de-n como las descritas en la sección 2.2.1).

El protocolo de firma distribuida que se propone a continuación sigue, fundamentalmente, lo establecido en [DD05]. La principal diferencia entre ambos protocolos está en la interactividad de los mismos. Mientras que la propuesta de Damgård y Dupont requiere la realización de pruebas de conocimiento interactivas, en el protocolo propuesto en esta memoria se eliminan dichas pruebas y se consigue una firma RSA distribuida totalmente no-interactiva.

A continuación se ofrece un esquema general del protocolo, se explican detalladamente las dos partes del mismo (Generación de la firma parcial y Reconstrucción de la firma) y se justifican los cambios introducidos.

3.2.1. Generación de la firma parcial

Un protocolo de firma de umbral (ver sección 2.4.1) suele componerse de dos subprotocolos. En el primero los participantes generan una firma parcial m_i del mensaje mientras que en el segundo se combinan dichas firmas parciales para obtener una firma válida del mensaje $Sign(m)$.

Las firmas parciales no deben revelar información alguna respecto a la clave privada ni deben permitir que ningún subconjunto no-autorizado de usuarios sea capaz de reconstruir la firma total. De hecho, las firmas parciales se realizan con la finalidad de que en ningún momento sea reconstruida la clave privada d, de manera que para firmar cada mensaje sea necesaria la colaboración de un subconjunto autorizado de usuarios[6].

En el protocolo propuesto en la figura 3.6 puede comprobarse que los participantes reciben un mensaje m y se limitan a calcular $m_i = m^{2\Delta d_i}$ (donde d_i es su fragmento de la clave privada d) y enviar dicha firma parcial al *reconstructor*.

Este procedimiento puede hacerse de manera totalmente independiente y asíncrona gracias a haber eliminado las pruebas de conocimiento que proponian Damgård y Dupond en su artículo. La eliminación de dichas pruebas no hace menos seguro ni menos robusto el protocolo aquí presentado dado que la única finalidad de éstas era reducir el

[6]Nos podríamos limitar a reconstruir la clave d y generar la firma m^d, que es un protocolo mucho más sencillo, pero en ese caso, aquel que obtuviera el valor de d podría generar cualquier firma sin la participación del resto de usuarios, que es algo que queremos evitar.

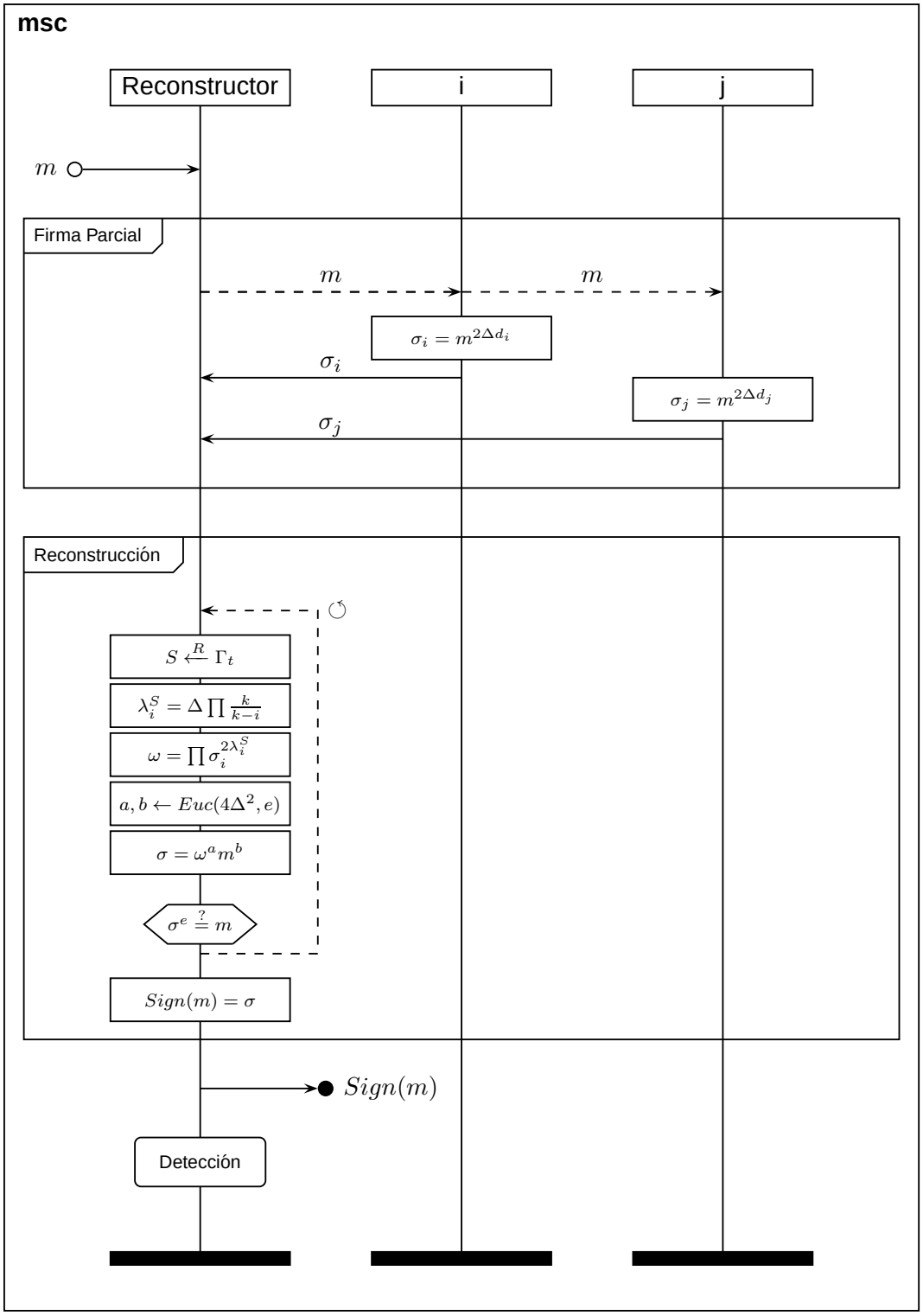


Figura 3.6: Protocolo de firma distribuida

número de reconstrucciones que es necesario realizar antes de obtener una firma válida, a una constante independiente del número de participantes. En la siguiente sección se explica por qué podemos obviar dichas pruebas sin poner en riesgo la robustez del protocolo y cómo, de hecho, se mejora la eficiencia del protocolo con esta medida.

3.2.2. Reconstrucción de la firma

La reconstrucción de la firma RSA distribuida puede llevarse a cabo por cualquier participante e, incluso, puede ser delegada en terceros sin que esto comprometa la seguridad del protocolo dado que es imposible obtener información alguna de la clave secreta a partir de las firmas parciales.

Para obtener una firma $Sign(m)$ a partir de las firmas parciales $\sigma_1, \ldots, \sigma_n$ es necesario escoger, de manera aleatoria y uniforme, un subgrupo de t participantes $S \in \Gamma_t$.

Acto seguido se calcularán los coeficientes de Lagrange correspondientes:

$$\lambda_i^S = \Delta \prod_{\substack{j \in S \\ j \neq i}} \frac{j}{j-i}$$

donde $\Delta = n!$.

Y se procederá a interpolar las firmas parciales:

$$\omega = \prod_{i \in S} \sigma_i^{2\lambda_i^S}$$

Si todas las firmas parciales σ_i han sido calculadas correctamente podemos asumir que $\omega = m^{4\Delta^2 d} \mod N$. Ahora bien, la firma que deseamos obtener es de la forma $m^d \mod N$ y por lo tanto aún tenemos que realizar los siguientes pasos.

Primero usamos el algoritmo de Euclides extendido para encontrar un par de valores a, b tales que $a \cdot 4\Delta^2 + b \cdot e = 1$. Nótese que esto tan sólo puede hacerse si $MCD(4\Delta^2, e) = 1$ pero dado que estamos trabajando con un número pequeño de participantes ($n \approx 10$) y que es habitual escoger como clave pública e un número primo de más de 16 bits, este requerimiento se cumple de manera automática.

Ahora sí, estamos en condiciones de obtener una firma global de m:

$$\sigma = \omega^a m^b \mod N = m^{a4\Delta^2 d} m^b \mod N$$

De nuevo, si todas las firmas parciales usadas en esta reconstrucción han sido correctamente calculadas es obvio que:

$$\sigma^e \mod N = m^{a4\Delta^2 de + be} \mod N = m^{a4\Delta^2 + be} \mod N$$

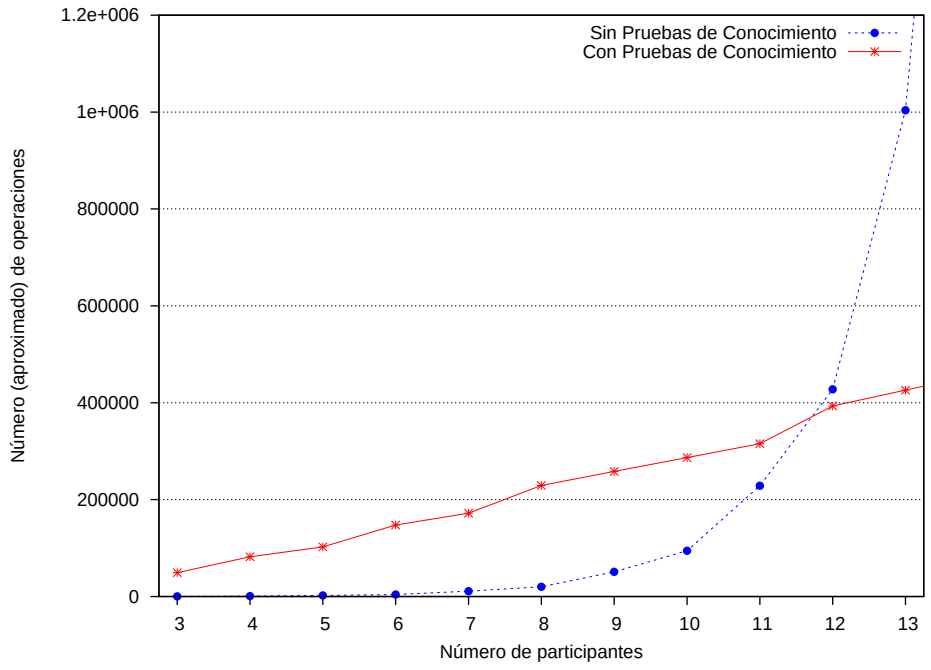
y por lo tanto ya hemos obtenido la firma global deseada. Sin embargo, es posible que en S haya algún usuario malicioso que haya mentido y por lo tanto es necesario comprobar que, efectivamente $\sigma^e = m \mod N$. En caso de que no se cumpla la anterior relación el protocolo de reconstrucción debe reiniciarse con un nuevo subgrupo de usuarios.

Como usamos la hipótesis de que hay, al menos, t usuarios que actúan honestamente es obvio que en algún momento conseguiremos la firma que estamos buscando. Lamentablemente, el número de reconstrucciones que será necesario realizar en el caso peor crece exponencialmente. Concretamente puede ser necesario realizar $\binom{n}{t} = \frac{n!}{t!(n-t)!}$, que para $n = 11$ y $t = 6$ es igual a 210.

Si bien lo expuesto en el párrafo anterior puede ser inaceptable en general, en el escenario particular al que se orienta este protocolo es, de hecho, más eficiente que la alternativa. En [DD05] se usan pruebas de conocimiento para reducir el número de reconstrucciones esperadas a un número constante que no depende de n ni de t. No obstante dichas pruebas de conocimiento son necesariamente interactivas y además tienen un gran coste computacional.

Así, si comparamos el volumen de cálculos que debe realizar el *reconstructor* en un caso y en otro, obtenemos que con pruebas de conocimiento se realizan del orden de unas $4096n\log_2(3t^t)$ operaciones por mensaje, mientras que sin ellas se realizan unas $45n\binom{n}{t}$. Es obvio que la segunda opción se vuelve inviable para valores grandes de n pero en el rango en el que nos movemos parece razonable usarla si, además, ganamos la no interactividad con ello.

Figura 3.7: Comparativa entre ambos métodos de reconstrucción



La figura 3.7 nos permite observar que el método propuesto en esta memoria es mucho más eficiente en el rango en el que se moverán los parámetros en el escenario de votaciones electrónicas (ver sección 1).

Además, en caso de que todos los participantes se comporten de manera honesta estaríamos hablando de una mejora de hasta 5 órdenes de magnitud si se omiten las pruebas de conocimiento. Este es, de hecho, el caso más frecuente y el que, con diferencia, dominará el cómputo global de tiempo de ejecución dado que cada vez que un usuario actúa de manera ilícita el protocolo es capaz de detectarlo y expulsarlo. En consecuencia, podemos esperar un comportamiento honesto la mayor parte del tiempo y tal suposición está en concordancia con el método aquí propuesto.

Es importante notar que obviar las pruebas de conocimiento no nos impide detectar y expulsar a los usuarios que mientan a la hora de realizar sus firmas parciales[7]. En efecto, una vez obtenida una firma válida, realizando tan sólo $n - t$ reconstrucciones más, podemos verificar las firmas parciales de todos los usuarios.

El subgrupo de usuarios S que ha permitido reconstruir la firma global está formado, obviamente, por t usuarios honestos. Por otra parte, para comprobar las firmas parciales de cada uno de los restantes $n-t$ usuarios tan sólo es necesario reconstruir la firma con las firmas parciales de $t - 1$ usuarios de S y la firma parcial del usuario que queramos poner a prueba. Si esta nueva reconstrucción es así mismo una firma global válida[8] sabremos que el nuevo usuario no ha mentido mientras que si el resultado no es válido sabremos que el usuario es malicioso y podremos expulsarlo.

[7] Esta es la otra función que tienen las pruebas de conocimiento en [DD05].

[8] De hecho es suficiente con comparar la ω generada por S y la $\tilde{\omega}$ generada por el nuevo subgrupo.

Capítulo 4

Conclusiones

Tras estudiar las diferentes propuestas teóricas, seleccionar las más adecuadas e implementarlas añadiendo las modificaciones necesarias para mejorar su rendimiento en el contexto para el que se ha realizado esta investigación, es hora de repasar los resultados obtenidos tanto a nivel teórico como a nivel práctico.

4.1. Resultados Teóricos

Si bien es cierto que a nivel teórico el problema de generar los parámetros y las firmas RSA de manera distribuida se daba por resuelto, no es menos cierto que en la literatura técnica se presta poca atención a la implementación práctica de los protocolos, descuidando así la eficiencia de los mismos.

También es cierto que las propuestas presentadas hasta el momento no se centraban en un escenario concreto y, bien al contrario, pretendían abarcar un amplio rango de aplicación, cosa que, de nuevo, mermaba su eficiencia.

A nivel teórico los principales resultados obtenidos durante este proyecto son:

- Se ha diseñado un protocolo integrado de generación de parámetros RSA y firma RSA distribuida orientado a procesos electorales.
- Dicho protocolo no hace uso de terceras partes de confianza ni de hipótesis no estándar.
- Dicho protocolo es robusto, seguro y completamente distribuido (siguiendo una estructura de acceso de umbral óptima[1])

[1] En este caso se entiende por óptima aquella que admite el máximo número de usuarios maliciosos, en particular cualquier minoría.

- Dicho protocolo es suficientemente eficiente como para ser usado en el contexto para el que ha sido diseñado usando hardware estándar.

4.2. Resultados Empíricos

Los resultados empíricos se han obtenido midiendo el tiempo que dedica a ejecutar el protocolo cada uno de los usuarios. A tal efecto se ha implentado el protocolo en Java y se han usado varios ordenadores en red local para simular los diferentes usuarios (y el Canal de Broadcast, como ya se explicó en la sección 2.1).

Generación de parámetros RSA

En lo que se refiere a la generación de parámetros RSA el dato que hay que tener en cuenta es el tiempo que tarda en ejecutarse una ronda consistente en la generación distribuida de N (ver sección 3.1.1) y el test de biprimalidad (ver sección 3.1.2). Esto es así dado que la tercera parte del protocolo, que es la generación de las claves e y d (ver sección 3.1.3), se ejecuta tan sólo una vez y equivale, aproximadamente a un par de dichas rondas.

	Tiempo por ronda (ms.)	
Tamaño de N (bits)	**No Robusto**	**Robusto**
256	320	2093
512	319	2927
1024	349	10330
2048	379	67510

Tabla 4.1: Generación de parámetros RSA con 3 usuarios

Los resultados obtenidos no son malos a pesar de que la implementación con la que se obtubieron dichos resultados no ha sido optimizada (pues se trataba de un prototipo) ni se ha usado en las pruebas hardware especializado.

Es de esperar que una implementación plenamente optimizada y, sobretodo, una red local que permita gestionar las conexiones con mayor rapidez puedan mejorarse los tiempos descritos en la tabla 4.1. En este sentido, cabe citar los resultados publicados por Malkin, Wu y Boneh, [MWB99] quienes consiguieron generar parámetros RSA de 2048 bits en poco más de 18 minutos usando una versión muy optimizada del protocolo no robusto de Boneh y Franklin.

Por otra parte, es fundamental recordar que dado que se trata de un algoritmo probabilístico del tipo *Las Vegas* no podemos predecir de antemano el número de rondas

que se necesitarán para obtener la respuesta correcta (que en este caso es un juego de parámetros RSA).

A pesar de ello, como el éxito del programa tan sólo depende de la capacidad de éste para encontrar una pareja de números primos de una determinada longitud podemos hacer estimaciones en función de resultados bien conocidos respecto a la densidad de los números primos en un determinado rango.

Así, si deseamos obtener un módulo RSA de tamaño 2κ (que sea producto de dos números primos de κ bits) el protocolo aquí presentado hará, de media, κ^2 rondas antes de finalizar.

Afortunadamente, con las mejoras introducidas en la sección 3.1.2 podemos suponer que todas esas rondas (a excepción de la última) se ejecutarán en tiempos comparables a los obtenidos con la versión no robusta del protocolo.

Firma Distribuida

En lo referente a la firma RSA (ver sección 3.2) los tiempos varían mucho en función de si se usa el método interactivo o el método no interactivo.

Paradójicamente, el método no interactivo es mucho más rápido que el interactivo debido a que las pruebas se hicieron con pocos usuarios (que es el escenario para el que se diseño esa versión del protocolo). Para entender mejor los detalles de ambas versiones del protocolo es necesario remitirse a la sección 3.2.2 (la figura 3.7 es especialmente reveladora).

	Tiempo por mensaje firmado (ms.)	
Tamaño de N (bits)	**No Interactivo**	**Interactivo**
256	~ 0	2350
512	~ 0	3650
1024	258	10850
2048	1040	59700

Tabla 4.2: Firma RSA con 3 usuarios

En cualquier caso, el proceso de firma es mucho más sencillo que el proceso de generación de parámetros y, además, si se usa la versión no interactiva del protocolo, es posible delegar la reconstrucción de la firma (ver sección 3.2.2) en terceras partes mientras los usuarios se dedican, exclusivamente, a realizar las firmas parciales (ver sección 3.2.1). El proceso podría, incluso, realizarse en diferido y tan sólo en caso de que se detectasen irregularidades en el recuento de votos (ver sección 1).

4.3. Posibles mejoras

A lo largo de esta memoria se han propuesto diversas mejoras sobre los protocolos de Frankel et al. [FMY98] y Damgård et al. [DD05]. En particular, cabe destacar el uso del escenario optimista en el protocolo de generación (ver sección 3.1.2) y la eliminación de las pruebas de conocimiento del protocolo de firma (ver sección 3.2.2). Ambas mejoras han sido tenidas en cuenta a la hora de presentar los resultados empíricos y han demostrado ser fundamentales para la aplicación práctica del protocolo de generación y firma RSA.

Por otra parte hay una serie de mejoras que no se han implementado pero que parecen interesantes de cara a optimizar el rendimiento o a proporcionar una mayor flexibilidad en su uso. En particular, el uso del *Replicated Integer Secret Sharing*, tal y como se describe en el apéndice A, permite implementar otras estructuras de acceso monótonas mejorando incluso el rendimiento del proceso de firma.

En lo que se refiere a optimizar la generación de parámetros, que es, sin duda, el proceso más costoso de este protocolo, es interesante estudiar las mejoras introducidas por Malkin, Wu y Boneh en el protocolo no robusto de Boneh y Franklin. Según afirman en su artículo, [MWB99], las mejoras más significativas que pueden aplicarse a la generación distribuida de parámetros RSA pasan por realizar test de divisibilidad sobre N antes de aplicarle el test de biprimalidad y por paralelizar el procedimiento en sí para aprovechar la capacidad computacional de los participantes que ahora se pierde mientras esperan la respuesta del resto de usuarios.

La primera mejora es obvia y fácil de implementar. Basta con que cada participante compruebe por su cuenta la divisibilidad de N por cierto número de primos de pequeño tamaño para eliminar rápidamente aquellos N que tengan factores pequeños y, en consecuencia, no sean válidos.

Otra manera de eliminar de antemano los módulos RSA con factores pequeños es asegurarse de que ni p ni q son divisibles por primos de pequeño tamaño. Lamentablemente ambos números están distribuidos y resulta fundamental para la seguridad del protocolo que permanezcan en secreto. Afortunadamente hay procedimientos, como el descrito en la sección $4{,}1$ de [MWB99], que permiten que cada participante genere sus fragmentos de p y q de manera que el resultado no sea divisible por primos de pequeño tamaño sin que de ello pueda deducirse información alguna sobre ambos números.

Finalmente, resulta útil que cada participante ejecute en paralelo varias copias del protocolo de generación de parámetros RSA de manera que se aprovechen los tiempos de espera entre que se envía un mensaje y se reciben los mensajes del resto de participantes. Esta mejora se basa en el hecho de que el cuello de botella del protocolo son las comunicaciones y no los cálculos. Malkin et al. comprobaron empíricamente que 6 ejecuciones en paralelo consiguen reducir a un tercio el tiempo necesario para obtener los parámetros.

Apéndice A

Replicated Integer Secret Sharing

A la hora de repartir la clave privada en un protocolo de firma de umbral es frecuente utilizar el esquema de compartición de secretos de Shamir (ver sección 2.2.2) o alguna de sus variantes (ver sección 2.2.3). Sin embargo, el de Shamir no es el único esquema de compartición de secretos que permite implementar una estructura de acceso de umbral (ver sección 2.2.1). A continuación se exponen las ventajas del esquema Replicated Integer Secret Sharing[1] en el contexto de las firmas de umbral en presencia de usuarios maliciosos.

A.1. Introducción

El RISS es un esquema de compartición de secretos muy flexible pero, en general, poco eficiente. En particular, se pueden implementar con dicho esquema todas las estructuras de acceso monótonas (ver sección 2.2.1) pero el número de fragmentos que cada usuario tendrá que guardar (y, en consecuencia, el número de firmas parciales que deberá realizar) es mucho mayor que en el caso del esquema de Shamir.

En particular, sea $\mathcal{C} = \{1, 2, \ldots, n\}$ el conjunto de usuarios entre los que se quiere compartir un secreto s y sea Γ el conjunto de subconjuntos de $\mathcal{C}$ que pueden reconstruir el secreto, definimos $\overline{\Gamma}$ como aquellos subconjuntos de $\mathcal{C}$ que no están autorizados a reconstruir el secreto. En particular $\Gamma \cup \overline{\Gamma} = \mathcal{P}(\mathcal{C})$ y $\Gamma \cap \overline{\Gamma} = \emptyset$.

Sea Υ el subconjunto de elementos maximales de $\overline{\Gamma}$, es decir, aquellos que no están contenidos en otro elemento de $\overline{\Gamma}$ y sea υ el cardinal de Υ. Entonces podemos implementar la estructura de acceso Γ mediante un esquema RISS en el que el secreto se reparte en υ fragmentos de manera aditiva ($s = s_1 + \cdots + s_\upsilon$) y cada participante recibe tantos fragmentos del mismo como elementos de Υ a los que no pertenece (recordemos

[1] En lo sucesivo: *RISS*

que Υ está formado por subconjuntos no autorizados maximales: $C \subset \mathcal{C}$ tal que $C \notin \Gamma$ y $\nexists C' \in \Upsilon$ tal que $C \subset C'$).

Para ello se asocia el fragmento $i-$ésimo del secreto s_i al elemento $i-$ésimo de Υ C_i de manera que todos los participantes que **no** pertenecen a C_i tendrán una copia de dicho fragmento del secreto. En particular cada participante posee varios fragmentos y cada fragmento es poseído por varios participantes.

Es un sencillo ejercicio comprobar que este esquema cumple realmente con su cometido (permite recuperar el secreto a todo los subconjuntos autorizados mientras que impide que los subconjuntos no autorizados tengan acceso a él).

A.1.1. RISS para estructuras de umbral

El el caso particular de las estructuras de umbral t-de-n tenemos que:

Parámetro	Shamir	RISS
Método de recuperación	Interpolación	Suma
Número de fragmentos	n	$\binom{n}{t-1}$
Fragmentos por participante	1	$\binom{n-1}{t-1}$
Participantes que tienen un mismo fragmento	1	$n-(t-1)$

De manera que, por cada mensaje que se desee firmar, los participantes deben generar $\binom{n-1}{t-1}$ firmas parciales (m^{s_i}) y enviarlas al *reconstructor*. Este último deberá elegir una firma parcial de cada tipo (habrá $\binom{n}{t-1}$ diferentes, y dispondrá de $n-(t-1)$ copias de cada una de ellas) y multiplicarlas entre sí para reconstruir la firma total:

$$\prod_{i=1}^{\binom{n}{t-1}} m^{s_i} = m^{\sum_i s_i} = m^s$$

A pesar de que el método de reconstrucción es mucho más sencillo que en el esquema de Shamir ($\binom{n}{t-1}-1$ productos contra n exponenciaciones modulares y $n-1$ productos), si comparamos ambos métodos resulta evidente que el RISS es mucho más ineficiente ($\binom{n-1}{t-1}$ firmas parciales por participante contra 1 firma parcial por participante). Esto es particularmente notable si $t \approx n/2$, punto en el que cada participante firmará un número exponencial de mensajes (exponencial respecto a n).

Sin embargo, resulta que pese a su ineficiencia es posible que el RISS sea la mejor opción a la hora de reconstruir la firma en presencia de usuarios maliciosos.

A.2. Detección de usuarios maliciosos mediante RISS

Recapitulemos, para reconstruir una firma usando el RISS es necesario que cada usuario envíe sus $\binom{n-1}{t-1}$ firmas parciales al *reconstructor* y que este último, que dispondrá de $n-(t-1)$ copias de cada firma parcial[2], seleccione una de cada y reconstruya la firma.

En el caso en el que todos los participantes son honestos el *reconstructor* cuenta con $n-(t-1)$ copias idénticas de cada firma parcial pero en presencia de usuarios maliciosos es posible que algunas de esas copias sean incorrectas y, por lo tanto, incoherente con el resto. Esto impediría, a priori, reconstruir la firma dado que en presencia de dos versiones de una misma firma parcial el *reconstructor* no sabría cuál escoger para realizar la reconstrucción.

Una posible solución consiste en probar todas las combinaciones posibles hasta obtener una firma parcial válida[3]. Pero llegados a este punto es necesario determinar cuál es el número máximo de reconstrucciones que será necesario realizar y comparar el coste computacional de este método con la alternativa de Shamir: $\binom{n}{t}$ reconstrucciones por interpolación en el caso peor.

Para reconstruir un secreto debemos usar una copia de cada firma parcial de manera que, a priori, podrían necesitarse $(n-(t-1))^{\binom{n}{t-1}}$ intentos. Sin embargo, también es suficiente con escoger el grupo de t usuarios cuyas firmas parciales se usarán en la reconstrucción, lo que reduce la cota superior a, *tan sólo*, $\binom{n}{t}$ intentos. Esta segunda cota mejora el resultado obtenido con Shamir dado que las reconstrucciones con el RISS son más eficientes pero, de hecho, es fácil ver que hacen falta muchos menos intentos.

En efecto, si escogemos un grupo de t usuarios tal que al menos dos usuarios del mismo hayan enviado copias incoherentes de una misma firma parcial, sabemos, de inmediato, que al menos uno de esos dos usuarios no es honesto y, por lo tanto, podemos ahorrarnos la reconstrucción correspondiente a ese grupo (que además estaría mal definida respecto a esa firma parcial). Dado que existen al menos t usuarios honestos sabemos que siempre habrá, al menos 1 grupo de t usuarios que es coherente. Además,

[2] Si todos los participantes actuasen de manera honesta sería suficiente con que t de ellos enviasen sus firmas parciales ya que en ese caso el *reconstructor* dispondría de, al menos, una copia de cada una de las $\binom{n-1}{t-1}$ firmas, pero en este caso usamos la estructura t-de-n para evitar que los usuarios maliciosos puedan boicotear el proceso de firma así que le pedimos a los n usuarios que envíen sus firmas parciales pero asumimos que algunos de ellos (como mucho $t-1$) pueden mentir en este punto.

[3] Pese a que el *reconstructor* no es capaz de distinguir una firma parcial correcta de una incorrecta, sí es capaz de determinar si la firma total es correcta o no, dado que conoce la clave pública.

si un grupo de $t+k$ usuarios es coherente en todas las firmas parciales podemos tratar como un único elemento a los $\binom{t+k}{t}$ subgrupos de t usuarios que hay en su interior en lo que a opciones de reconstrucción se refiere.

Llegados a este punto quizá convenga replantear el problema usando terminología de la Teoría de Grafos:

Sea $G = \{V, E\}$ el grafo cuyos conjunto de vértices representa a los diferentes usuarios ($V = \{1, 2, \ldots, n\}$) y cuyas aristas representan la coherencia entre pares de usuarios ($E = \{(i, j) \mid i$ es coherente con j en todas las firmas parciales $\}$), entonces el número máximo de reconstrucciones que hay que hacer hasta conseguir una firma correcta es igual al número de $t-$cliques maximales del G, siendo un $t-$clique maximal H un subgrafo de G que cumple las siguientes condiciones:

Coherencia: El subgrafo debe ser completo, es decir, $i, j \in H \Rightarrow (i, j) \in E$.

Maximalidad: El subgrafo no está contenido en ningún otro $t-$clique maximal.

Tamaño: El subgrafo contiene, al menos, t vértices de G.

En particular, si todos los usuarios son honestos tan sólo existe un $t-$clique maximal, que es el propio G y, por lo tanto, tan sólo es necesaria una reconstrucción para obtener la firma. Por otra parte, si los $t-1$ usuarios maliciosos mienten en todas sus respuestas resulta que también existe un único $t-$clique maximal, el formado por los t usuarios honestos.

Desde esta perspectiva es fácil realizar algunas observaciones que ilustran cómo el RISS ayuda a detectar rápidamente los comportamientos maliciosos[4]:

Los usuarios maliciosos mentirán poco: La mejor estrategia para los usuarios maliciosos no puede ser mentir en toda ocasión ya que, como se ha visto, esto reduce el número de $t-$cliques maximales. En particular, todo vértice de G cuyo grado sea inferior a $t-1$ puede ser eliminado directamente (simplificando así el problema) y, por lo tanto, los usuarios maliciosos deben ser coherentes con, al menos, otros $t-1$ usuarios. Pese a que en este grupo de $t-1$ usuarios puede haber tanto usuarios honestos como usuarios maliciosos, es obvio que su capacidad de maniobra es muy reducida.

Si todas las copias de una firma parcial son coherentes, todas con correctas: Esto es obvio dado que entre todos los usuarios honestos se genera al menos una copia de cada firma parcial. Así pues, si todas las copias son coherentes entre sí, en particular lo son con la del usuario honesto, que sabemos que es correcta.

[4] Nótese que la estructura del grafo G puede ser reconstruida y estudiada por el *reconstructor* en tiempo $O(n^2)$ simplemente comparando las firmas parciales recibidas.

La mejor estrategia supone, como mínimo, t reconstrucciones: La mejor estrategia para los usuarios maliciosos debe ser mejor o igual a la que se describe a continuación: supongamos que los usuarios maliciosos dicen la verdad en todos los casos excepto en una de las firmas parciales que tienen en común. En dicha firma parcial generan $t-1$ respuestas diferentes entre sí y diferentes a la respuesta correcta que generará el usuario honesto que también participa en dicha firma parcial. Esto nos deja con t opciones indistinguibles a priori.

$$t \quad \leqslant \quad \#\{t-cliques\ maximales\ de\ G\} \quad \leqslant \quad \binom{n}{t}$$

En referencia a la última observación, es razonable conjeturar que la cota inferior dada es óptima. De hecho, esto se cumple para los casos $n = 3, 5, 7$ y 9, como se ha podido comprobar computacionalmente. Lamentablemente no se ha encontrado ningún resultado en la literatura, ni se ha podido demostrar de manera concluyente que esto es así para todo n.

Sin embargo, y dado que la cota superior iguala a la obtenida con el esquema de Shamir podemos concluir que el RISS es más eficiente desde el punto de vista del *reconstructor* (a expensas de ser menos eficiente desde el punto de vista de los firmantes y desde el punto de vista de las comunicaciones).

Se trata pues de un método ideal cuando la capacidad computacional del *reconstructor* sea muy limitada en comparación con la de los firmantes (por usar un hardware poco potente o por tener que reconstruir firmas de muchos grupos de firmantes). Por el contrario, es un método poco recomendable en caso de que los firmantes no dispongan de mucha capacidad computacional o de que las comunicaciones resulten costosas.

Finalmente es necesario remarcar que el RISS es un esquema mucho más flexible que el esquema de Shamir y que permite implementar otras estructuras de acceso monótonas aparte de los umbrales. Además, se pueden conseguir mejoras sustanciales en la eficiencia del esquema RISS si asumimos la presencia de pocos usuarios maliciosos ($n - t \ll n/2$).

Bibliografía

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Yung [Yun02], pages 417–432. *(Citado en las páginas 7 y 8)*

[BF97] Dan Boneh and Matthew K. Franklin. Efficient generation of shared rsa keys (extended abstract). In Kaliski [Kal97], pages 425–439. *(Citado en las páginas 6, 33, 37 y 41)*

[BTHR07] Zuzana Beerliová-Trubíniová, Martin Hirt, and Micha Riser. Efficient byzantine agreement with faulty minority. In Kurosawa [Kur07], pages 393–409. *(Citado en la página 16)*

[DBL99] *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*. The Internet Society, 1999. *(Citado en la página 60)*

[DD05] Ivan Damgård and Kasper Dupont. Efficient threshold rsa signatures with general moduli and no extra assumptions. In Vaudenay [Vau05], pages 346–361. *(Citado en las páginas 8, 20, 21, 44, 47, 48 y 52)*

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Zheng [Zhe02], pages 125–142. *(Citado en las páginas 23 y 25)*

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In Pfitzmann [Pfi01], pages 152–165. *(Citado en la página 8)*

[DK02] Hans Delfs and Helmut Knebl. *Introduction to Cryptography: Principles and Applications*. Springer, 2002. *(Citado en la página 37)*

[DM10] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed rsa key generation. In Micciancio [Mic10], pages 183–200. *(Citado en la página 7)*

[Fei92] Joan Feigenbaum, editor. *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992. *(Citado en la página 61)*

[FGMY97] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Optimal resilience proactive public-key cryptosystems. In *FOCS*, pages 384–393, 1997. *(Citado en la página 20)*

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed rsa-key generation. In *PODC*, page 320, 1998. *(Citado en las páginas 6, 33, 37, 41 y 52)*

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko [Odl87], pages 186–194. *(Citado en la página 40)*

[GJKR01] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. *Inf. Comput.*, 164(1):54–84, 2001. *(Citado en la página 20)*

[Kal97] Burton S. Kaliski, editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997. *(Citado en la página 59)*

[Kur07] Kaoru Kurosawa, editor. *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*. Springer, 2007. *(Citado en la página 59)*

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. *(Citado en la página 16)*

[MB01] Sjouke Mauw and Victor Bos. Drawing Message Sequence Charts with LaTeX. *TUGBoat*, 22(1-2):87–92, March/June 2001. *(Citado en la página 9)*

[Mic10] Daniele Micciancio, editor. *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*. Springer, 2010. *(Citado en la página 59)*

[MWB99] Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of rsa keys. In *NDSS* [DBL99]. *(Citado en las páginas 50 y 52)*

[Odl87] Andrew M. Odlyzko, editor. *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*. Springer, 1987. *(Citado en la página 60)*

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum [Fei92], pages 129–140. *(Citado en la página 23)*

[Pfi01] Birgit Pfitzmann, editor. *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001. *(Citado en la página 59)*

[Pre00] Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000. *(Citado en la página 61)*

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. *(Citado en la página 29)*

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. *(Citado en la página 17)*

[Sho00] Victor Shoup. Practical threshold signatures. In Preneel [Pre00], pages 207–220. *(Citado en la página 8)*

[Vau05] Serge Vaudenay, editor. *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*. Springer, 2005. *(Citado en la página 59)*

[Yun02] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002. *(Citado en la página 59)*

[Zhe02] Yuliang Zheng, editor. *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*. Springer, 2002. *(Citado en la página 59)*